

CPU hotplug Support in Linux(tm) Kernel

Maintainers:

CPU Hotplug Core:

Rusty Russell <rusty@rustycorp.com.au>

Srivatsa Vaddagiri <vatsa@in.ibm.com>

i386:

Zwane Mwaikambo <zwane@arm.linux.org.uk>

ppc64:

Nathan Lynch <nathanl@austin.ibm.com>

Joel Schopp <jschopp@austin.ibm.com>

ia64/x86\_64:

Ashok Raj <ashok.raj@intel.com>

s390:

Heiko Carstens <heiko.carstens@de.ibm.com>

Authors: Ashok Raj <ashok.raj@intel.com>

Lots of feedback: Nathan Lynch <nathanl@austin.ibm.com>,

Joel Schopp <jschopp@austin.ibm.com>

Introduction

Modern advances in system architectures have introduced advanced error reporting and correction capabilities in processors. CPU architectures permit partitioning support, where compute resources of a single CPU could be made available to virtual machine environments. There are couple OEMS that support NUMA hardware which are hot pluggable as well, where physical node insertion and removal require support for CPU hotplug.

Such advances require CPUs available to a kernel to be removed either for provisioning reasons, or for RAS purposes to keep an offending CPU off system execution path. Hence the need for CPU hotplug support in the Linux kernel.

A more novel use of CPU-hotplug support is its use today in suspend resume support for SMP. Dual-core and HT support makes even a laptop run SMP kernels which didn't support these methods. SMP support for suspend/resume is a work in progress.

General Stuff about CPU Hotplug

-----

## Command Line Switches

-----  
maxcpus=n      Restrict boot time cpus to n. Say if you have 4 cpus, using  
                 maxcpus=2 will only boot 2. You can choose to bring the  
                 other cpus later online, read FAQ's for more info.

additional\_cpus=n (\*) Use this to limit hotpluggable cpus. This option sets  
                      cpu\_possible\_map = cpu\_present\_map + additional\_cpus

cede\_offline={"off","on"} Use this option to disable/enable putting offlined  
                      processors to an extended H\_CEDDE state on  
                      supported pseries platforms.  
                      If nothing is specified,  
                      cede\_offline is set to "on".

(\*) Option valid only for following architectures

- ia64

ia64 uses the number of disabled local apics in ACPI tables MADT to  
determine the number of potentially hot-pluggable cpus. The implementation  
should only rely on this to count the # of cpus, but \*MUST\* not rely  
on the apicid values in those tables for disabled apics. In the event  
BIOS doesn't mark such hot-pluggable cpus as disabled entries, one could  
use this parameter "additional\_cpus=x" to represent those cpus in the  
cpu\_possible\_map.

possible\_cpus=n    [s390,x86\_64] use this to set hotpluggable cpus.  
                      This option sets possible\_cpus bits in  
                      cpu\_possible\_map. Thus keeping the numbers of bits set  
                      constant even if the machine gets rebooted.

CPU maps and such

-----  
[More on cpumaps and primitive to manipulate, please check  
include/linux/cpumask.h that has more descriptive text.]

cpu\_possible\_map: Bitmap of possible CPUs that can ever be available in the  
system. This is used to allocate some boot time memory for per\_cpu variables  
that aren't designed to grow/shrink as CPUs are made available or removed.  
Once set during boot time discovery phase, the map is static, i.e no bits

are added or removed anytime. Trimming it accurately for your system needs upfront can save some boot time memory. See below for how we use heuristics in x86\_64 case to keep this under check.

`cpu_online_map`: Bitmap of all CPUs currently online. Its set in `__cpu_up()` after a cpu is available for kernel scheduling and ready to receive interrupts from devices. Its cleared when a cpu is brought down using `__cpu_disable()`, before which all OS services including interrupts are migrated to another target CPU.

`cpu_present_map`: Bitmap of CPUs currently present in the system. Not all of them may be online. When physical hotplug is processed by the relevant subsystem (e.g ACPI) can change and new bit either be added or removed from the map depending on the event is hot-add/hot-remove. There are currently no locking rules as of now. Typical usage is to init topology during boot, at which time hotplug is disabled.

You really dont need to manipulate any of the system cpu maps. They should be read-only for most use. When setting up per-cpu resources almost always use `cpu_possible_map/for_each_possible_cpu()` to iterate.

Never use anything other than `cpumask_t` to represent bitmap of CPUs.

```
#include <linux/cpumask.h>
```

```
for_each_possible_cpu    - Iterate over cpu_possible_map
for_each_online_cpu      - Iterate over cpu_online_map
for_each_present_cpu     - Iterate over cpu_present_map
for_each_cpu_mask(x,mask) - Iterate over some random collection of cpu mask.
```

```
#include <linux/cpu.h>
```

```
get_online_cpus() and put_online_cpus():
```

The above calls are used to inhibit cpu hotplug operations. While the `cpu_hotplug.refcount` is non zero, the `cpu_online_map` will not change. If you merely need to avoid cpus going away, you could also use `preempt_disable()` and `preempt_enable()` for those sections. Just remember the critical section cannot call any function that can sleep or schedule this process away. The `preempt_disable()` will work as long as `stop_machine_run()` is used to take a cpu down.

CPU Hotplug - Frequently Asked Questions.

Q: How to enable my kernel to support CPU hotplug?

A: When doing make defconfig, Enable CPU hotplug support

```
"Processor type and Features" -> Support for Hotpluggable CPUs
```

Make sure that you have CONFIG\_HOTPLUG, and CONFIG\_SMP turned on as well.

You would need to enable CONFIG\_HOTPLUG\_CPU for SMP suspend/resume support as well.

Q: What architectures support CPU hotplug?

A: As of 2.6.14, the following architectures support CPU hotplug.

```
i386 (Intel), ppc, ppc64, parisc, s390, ia64 and x86_64
```

Q: How to test if hotplug is supported on the newly built kernel?

A: You should now notice an entry in sysfs.

Check if sysfs is mounted, using the "mount" command. You should notice an entry as shown below in the output.

```
....  
none on /sys type sysfs (rw)  
....
```

If this is not mounted, do the following.

```
#mkdir /sysfs  
#mount -t sysfs sys /sys
```

Now you should see entries for all present cpu, the following is an example in a 8-way system.

```
#pwd  
#/sys/devices/system/cpu  
#ls -l  
total 0  
drwxr-xr-x 10 root root 0 Sep 19 07:44 .  
drwxr-xr-x 13 root root 0 Sep 19 07:45 ..
```

```
drwxr-xr-x  3 root root 0 Sep 19 07:44 cpu0
drwxr-xr-x  3 root root 0 Sep 19 07:44 cpu1
drwxr-xr-x  3 root root 0 Sep 19 07:44 cpu2
drwxr-xr-x  3 root root 0 Sep 19 07:44 cpu3
drwxr-xr-x  3 root root 0 Sep 19 07:44 cpu4
drwxr-xr-x  3 root root 0 Sep 19 07:44 cpu5
drwxr-xr-x  3 root root 0 Sep 19 07:44 cpu6
drwxr-xr-x  3 root root 0 Sep 19 07:48 cpu7
```

Under each directory you would find an "online" file which is the control file to logically online/offline a processor.

Q: Does hot-add/hot-remove refer to physical add/remove of cpus?

A: The usage of hot-add/remove may not be very consistently used in the code. CONFIG\_HOTPLUG\_CPU enables logical online/offline capability in the kernel. To support physical addition/removal, one would need some BIOS hooks and the platform should have something like an attention button in PCI hotplug. CONFIG\_ACPI\_HOTPLUG\_CPU enables ACPI support for physical add/remove of CPUs.

Q: How do i logically offline a CPU?

A: Do the following.

```
#echo 0 > /sys/devices/system/cpu/cpuX/online
```

Once the logical offline is successful, check

```
#cat /proc/interrupts
```

You should now not see the CPU that you removed. Also online file will report the state as 0 when a cpu is offline and 1 when its online.

```
#To display the current cpu state.
```

```
#cat /sys/devices/system/cpu/cpuX/online
```

Q: Why cant i remove CPU0 on some systems?

A: Some architectures may have some special dependency on a certain CPU.

For e.g in IA64 platforms we have ability to sent platform interrupts to the OS. a.k.a Corrected Platform Error Interrupts (CPEI). In current ACPI specifications, we didn't have a way to change the target CPU. Hence if the current ACPI version doesn't support such re-direction, we disable that CPU

by making it not-removable.

In such cases you will also notice that the online file is missing under cpu0.

Q: How do i find out if a particular CPU is not removable?

A: Depending on the implementation, some architectures may show this by the absence of the "online" file. This is done if it can be determined ahead of time that this CPU cannot be removed.

In some situations, this can be a run time check, i.e if you try to remove the last CPU, this will not be permitted. You can find such failures by investigating the return value of the "echo" command.

Q: What happens when a CPU is being logically offlined?

A: The following happen, listed in no particular order :-)

- A notification is sent to in-kernel registered modules by sending an event CPU\_DOWN\_PREPARE or CPU\_DOWN\_PREPARE\_FROZEN, depending on whether or not the CPU is being offlined while tasks are frozen due to a suspend operation in progress
- All processes are migrated away from this outgoing CPU to new CPUs. The new CPU is chosen from each process' current cpuset, which may be a subset of all online CPUs.
- All interrupts targeted to this CPU is migrated to a new CPU
- timers/bottom half/task lets are also migrated to a new CPU
- Once all services are migrated, kernel calls an arch specific routine \_\_cpu\_disable() to perform arch specific cleanup.
- Once this is successful, an event for successful cleanup is sent by an event CPU\_DEAD (or CPU\_DEAD\_FROZEN if tasks are frozen due to a suspend while the CPU is being offlined).

"It is expected that each service cleans up when the CPU\_DOWN\_PREPARE notifier is called, when CPU\_DEAD is called its expected there is nothing running on behalf of this CPU that was offlined"

Q: If i have some kernel code that needs to be aware of CPU arrival and departure, how to i arrange for proper notification?

A: This is what you would need in your kernel code to receive notifications.

```
#include <linux/cpu.h>
static int __cpuinit foobar_cpu_callback(struct notifier_block *nfb,
```

```
        unsigned long action, void *hcpu)
{
    unsigned int cpu = (unsigned long)hcpu;

    switch (action) {
    case CPU_ONLINE:
    case CPU_ONLINE_FROZEN:
        foobar_online_action(cpu);
        break;
    case CPU_DEAD:
    case CPU_DEAD_FROZEN:
        foobar_dead_action(cpu);
        break;
    }
    return NOTIFY_OK;
}

static struct notifier_block __cpuinitdata foobar_cpu_notifier =
{
    .notifier_call = foobar_cpu_callback,
};
```

You need to call `register_cpu_notifier()` from your init function.

Init functions could be of two types:

1. early init (init function called when only the boot processor is online).
2. late init (init function called `_after_` all the CPUs are online).

For the first case, you should add the following to your init function

```
register_cpu_notifier(&foobar_cpu_notifier);
```

For the second case, you should add the following to your init function

```
register_hotcpu_notifier(&foobar_cpu_notifier);
```

You can fail PREPARE notifiers if something doesn't work to prepare resources.

This will stop the activity and send a following CANCELED event back.

CPU\_DEAD should not be failed, its just a goodness indication, but bad things will happen if a notifier in path sent a BAD notify code.

Q: I don't see my action being called for all CPUs already up and running?

A: Yes, CPU notifiers are called only when new CPUs are on-lined or offlined.

If you need to perform some action for each cpu already in the system, then

```
for_each_online_cpu(i) {
    foobar_cpu_callback(&foobar_cpu_notifier, CPU_UP_PREPARE, i);
    foobar_cpu_callback(&foobar_cpu_notifier, CPU_ONLINE, i);
}
```

Q: If i would like to develop cpu hotplug support for a new architecture, what do i need at a minimum?

A: The following are what is required for CPU hotplug infrastructure to work correctly.

- Make sure you have an entry in Kconfig to enable CONFIG\_HOTPLUG\_CPU
- \_\_cpu\_up() - Arch interface to bring up a CPU
- \_\_cpu\_disable() - Arch interface to shutdown a CPU, no more interrupts can be handled by the kernel after the routine returns. Including local APIC timers etc are shutdown.
- \_\_cpu\_die() - This actually supposed to ensure death of the CPU. Actually look at some example code in other arch that implement CPU hotplug. The processor is taken down from the idle() loop for that specific architecture. \_\_cpu\_die() typically waits for some per\_cpu state to be set, to ensure the processor dead routine is called to be sure positively.

Q: I need to ensure that a particular cpu is not removed when there is some work specific to this cpu is in progress.

A: There are two ways. If your code can be run in interrupt context, use `smp_call_function_single()`, otherwise use `work_on_cpu()`. Note that `work_on_cpu()` is slow, and can fail due to out of memory:

```
int my_func_on_cpu(int cpu)
{
    int err;
    get_online_cpus();
    if (!cpu_online(cpu))
        err = -EINVAL;
    else
```

```
#if NEEDS_BLOCKING
    err = work_on_cpu(cpu, __my_func_on_cpu, NULL);
#else
    smp_call_function_single(cpu, __my_func_on_cpu, &err,
        true);
#endif
    put_online_cpus();
    return err;
}
```

Q: How do we determine how many CPUs are available for hotplug.

A: There is no clear spec defined way from ACPI that can give us that information today. Based on some input from Natalie of Unisys, that the ACPI MADT (Multiple APIC Description Tables) marks those possible CPUs in a system with disabled status.

Andi implemented some simple heuristics that count the number of disabled CPUs in MADT as hotpluggable CPUS. In the case there are no disabled CPUS we assume 1/2 the number of CPUs currently present can be hotplugged.

Caveat: Today's ACPI MADT can only provide 256 entries since the apicid field in MADT is only 8 bits.

#### User Space Notification

Hotplug support for devices is common in Linux today. Its being used today to support automatic configuration of network, usb and pci devices. A hotplug event can be used to invoke an agent script to perform the configuration task.

You can add /etc/hotplug/cpu.agent to handle hotplug notification user space scripts.

```
#!/bin/bash
# $Id: cpu.agent
# Kernel hotplug params include:
#ACTION=%s [online or offline]
#DEVPATH=%s
#
cd /etc/hotplug
. ./hotplug.functions
```

```
case $ACTION in
online)
    echo `date` ":cpu.agent" add cpu >> /tmp/hotplug.txt
    ;;
offline)
    echo `date` ":cpu.agent" remove cpu >>/tmp/hotplug.txt
    ;;
*)
    debug_mesg CPU $ACTION event not supported
    exit 1
    ;;
esac
```