Paravirt_ops on IA64

21 May 2008, Isaku Yamahata <yamahata@valinux.co.jp>

Introduction

The aim of this documentation is to help with maintainability and/or to encourage people to use paravirt_ops/IA64.

paravirt_ops (pv_ops in short) is a way for virtualization support of Linux kernel on x86. Several ways for virtualization support were proposed, paravirt_ops is the winner.
On the other hand, now there are also several IA64 virtualization technologies like kvm/IA64, xen/IA64 and many other academic IA64 hypervisors so that it is good to add generic virtualization infrastructure on Linux/IA64.

What is paravirt_ops?

It has been developed on x86 as virtualization support via API, not ABI. It allows each hypervisor to override operations which are important for hypervisors at API level. And it allows a single kernel binary to run on all supported execution environments including native machine.
Essentially paravirt_ops is a set of function pointers which represent operations corresponding to low level sensitive instructions and high level functionalities in various area. But one significant difference from usual function pointer table is that it allows optimization with binary patch. It is because some of these operations are very performance sensitive and indirect call overhead is not negligible. With binary patch, indirect C function call can be transformed into direct C function call or in-place execution to eliminate the overhead.

Thus, operations of paravirt_ops are classified into three categories.

- simple indirect call
  These operations correspond to high level functionality so that the overhead of indirect call isn't very important.

- indirect call which allows optimization with binary patch
  Usually these operations correspond to low level instructions. They are called frequently and performance critical. So the overhead is very important.

- a set of macros for hand written assembly code
  Hand written assembly codes (.S files) also need paravirtualization because they include sensitive instructions or some of code paths in them are very performance critical.

The relation to the IA64 machine vector

Linux/IA64 has the IA64 machine vector functionality which allows the kernel to switch implementations (e.g. initialization, ipi, dma api...) depending on executing platform.
We can replace some implementations very easily defining a new machine

```
vector. Thus another approach for virtualization support would be
enhancing the machine vector functionality.
But paravirt_ops approach was taken because
- virtualization support needs wider support than machine vector does.
  e.g. low level instruction paravirtualization. It must be
       initialized very early before platform detection.

- virtualization support needs more functionality like binary patch.
  Probably the calling overhead might not be very large compared to the
  emulation overhead of virtualization. However in the native case, the
  overhead should be eliminated completely.
  A single kernel binary should run on each environment including native,
  and the overhead of paravirt_ops on native environment should be as
  small as possible.

- for full virtualization technology, e.g. KVM/IA64 or
  Xen/IA64 HVM domain, the result would be
  (the emulated platform machine vector. probably dig) + (pv_ops).
  This means that the virtualization support layer should be under
  the machine vector layer.

Possibly it might be better to move some function pointers from
paravirt_ops to machine vector. In fact, Xen domU case utilizes both
pv_ops and machine vector.


IA64 paravirt_ops
-----------------
In this section, the concrete paravirt_ops will be discussed.
Because of the architecture difference between ia64 and x86, the
resulting set of functions is very different from x86 pv_ops.

- C function pointer tables
They are not very performance critical so that simple C indirect
function call is acceptable. The following structures are defined at
this moment. For details see linux/include/asm-ia64/paravirt.h
  - struct pv_info
    This structure describes the execution environment.
  - struct pv_init_ops
    This structure describes the various initialization hooks.
  - struct pv_iosapic_ops
    This structure describes hooks to iosapic operations.
  - struct pv_irq_ops
    This structure describes hooks to irq related operations
  - struct pv_time_op
    This structure describes hooks to steal time accounting.

- a set of indirect calls which need optimization
Currently this class of functions correspond to a subset of IA64
intrinsics. At this moment the optimization with binary patch isn't
implemented yet.
struct pv_cpu_op is defined. For details see
linux/include/asm-ia64/paravirt_privop.h
Mostly they correspond to ia64 intrinsics 1-to-1.
Caveat: Now they are defined as C indirect function pointers, but in
order to support binary patch optimization, they will be changed
```

using GCC extended inline assembly code.

- a set of macros for hand written assembly code (.S files)
For maintenance purpose, the taken approach for .S files is single
source code and compile multiple times with different macros definitions.
Each pv_ops instance must define those macros to compile.
The important thing here is that sensitive, but non-privileged
instructions must be paravirtualized and that some privileged
instructions also need paravirtualization for reasonable performance.
Developers who modify .S files must be aware of that. At this moment
an easy checker is implemented to detect paravirtualization breakage.
But it doesn't cover all the cases.

Sometimes this set of macros is called pv_cpu_asm_op. But there is no
corresponding structure in the source code.
Those macros mostly 1:1 correspond to a subset of privileged
instructions. See linux/include/asm-ia64/native/inst.h.
And some functions written in assembly also need to be overrided so
that each pv_ops instance have to define some macros. Again see
linux/include/asm-ia64/native/inst.h.

Those structures must be initialized very early before start_kernel.
Probably initialized in head.S using multi entry point or some other trick.
For native case implementation see linux/arch/ia64/kernel/paravirt.c.