Subject: [PATCH RFC 1/31] net: Add net_namespace_type.h to allow for per network namespace variables.
Posted by ebiederm on Thu, 25 Jan 2007 19:00:03 GMT
View Forum Message <> Reply to Message

The problem:
To properly implement a ``level 2'' network namespace we need to move many of the networking stack global variables into the network namespace. We want to keep it explicit that the code is accessing a variable in a network namespace. We want to be able to completely compile out the network namespace support so we can do comparitive performance testing, and so to not penalize users who don't need network namespace support. Because the network stack is a moving target we want something simple that allows for the bulk of the changes to be merged before we enable network namespace support.

My biggest challenge when looking into this was to find an approach that would allow the code to compile out, in a way that does not yield any performance overhead and does not make the code ugly. While playing with the different possibilities I discovered that gcc will not pass 0 byte structures that are arguments to functions and instead will simply optmize them away. This appears to be true on i386 all of the way back to gcc-2.95 and I verified that it also works with gcc 4.1 on x86_64. Since this is part of the ABI I never expect it to change. Hopefully gcc uses this nice optimization on all architectures, I suspect so as C++ allows passing function arguments of type void in certain circumstances.

Using this observation I was able to come up with an network namespace implementation network namespace code that allows the changes to completely compile out when we don't build the kernel with network namespace support.

This patch implements my dummy network namespace support that should completely compiles out. Further patches will add the real version. Starting with the dummy gives a quick hint of where I am going and allows for dependencies to be overcome.

When doing my proof of concept implementation one of the other problems I had was that as the network stack comes in so many modular pieces figuring out how to get their global variables into the network namespace structure was a challenge. The basic technique used by our per cpu variables for having the linker build and dynamically change structures for us appears applicable here and a lot less nuisance then what I did before so I am implementing a tailored version of that technique as well, and again this makes it very simple to compile the code out.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 include/linux/net_namespace_type.h | 52 ++++++++++++++++++++++++++++++++++++
 1 files changed, 52 insertions(+), 0 deletions(-)

diff --git a/include/linux/net_namespace_type.h b/include/linux/net_namespace_type.h
new file mode 100644
index 0000000..8173f59
--- /dev/null
+++ b/include/linux/net_namespace_type.h
@@ -0,0 +1,52 @@
+/*
+ * Definition of the network namespace reference type
+ * And operations upon it.
+ */
+#ifndef __LINUX_NET_NAMESPACE_TYPE_H
+#define __LINUX_NET_NAMESPACE_TYPE_H
+
+#define __pernetname(name) per_net__##name
+
+typedef struct {} net_t;
+
+#define __data_pernet
+
+/* Look up a per network namespace variable */
+static inline unsigned long __per_net_offset(net_t net) { return 0; }
+
+/* Like per_net but returns a pseudo variable address that must be moved
+ * __per_net_offset() bytes before it will point to a real variable.
+ * Useful for static initializers.
+ */
+#define __per_net_base(name)   __pernetname(name)
+
+/* Get the network namespace reference from a per_net variable address */
+#define net_of(ptr, name) ({ net_t net; ptr; net; })
+
+/* Look up a per network namespace variable */
+#define per_net(name, net) \
+ (*(__per_net_offset(net), &__per_net_base(name)))
+
+/* Are the two network namespaces the same */
+static inline int net_eq(net_t a, net_t b) { return 1; }
+/* Get an unsigned value appropriate for hashing the network namespace */
+static inline unsigned int net_hval(net_t net) { return 0; }
+
+/* Convert to and from to and from void pointers */
+static inline void *net_to_voidp(net_t net) { return NULL; }
+static inline net_t net_from_voidp(void *ptr) { net_t net; return net; }

```
+
+static inline int null_net(net_t net) { return 0; }
+
+#define DEFINE_PER_NET(type, name)    \
+ __data_pernet __typeof__(type) __pernetname(name)
+
+#define DECLARE_PER_NET(type, name) \
+ extern __typeof__(type) __pernetname(name)
+
+#define EXPORT_PER_NET_SYMBOL(var) \
+ EXPORT_SYMBOL(__pernetname(var))
+#define EXPORT_PER_NET_SYMBOL_GPL(var) \
+ EXPORT_SYMBOL_GPL(__pernetname(var))
+
+#endif /* __LINUX_NET_NAMESPACE_TYPE_H */
--
1.4.4.1.g278f
```