Subject: [RFC PATCH 0/5] net: socket bind to file descriptor introduced
Posted by Stanislav Kinsbursky on Wed, 15 Aug 2012 16:21:56 GMT
View Forum Message <> Reply to Message

This patch set introduces new socket operation and new system call:
sys_fbind(), which allows to bind socket to opened file.
File to bind to can be created by sys_mknod(S_IFSOCK) and opened by
open(O_PATH).

This system call is especially required for UNIX sockets, which has name
lenght limitation.

The following series implements...

Stanislav Kinsbursky (5):
net: cleanup unix_bind() a little
net: split unix_bind()
net: new protocol operation fbind() introduced
net: fbind() for unix sockets protocol operations introduced
syscall: sys_fbind() introduced

arch/x86/syscalls/syscall_32.tbl | 1
arch/x86/syscalls/syscall_64.tbl | 1
include/linux/net.h | 2 +
include/linux/syscalls.h | 1
kernel/sys_ni.c | 3 +
net/socket.c | 25 +++++++
net/unix/af_unix.c | 136 ++++++++++++++++++++++++++++++++--------
7 files changed, 140 insertions(+), 29 deletions(-)

Subject: [RFC PATCH 1/5] net: cleanup unix_bind() a little
Posted by Stanislav Kinsbursky on Wed, 15 Aug 2012 16:22:04 GMT
View Forum Message <> Reply to Message

This will simplify further changes for unix_fbind().
---
net/unix/af_unix.c | 12 +++++-------
1 files changed, 5 insertions(+), 7 deletions(-)

diff --git a/net/unix/af_unix.c b/net/unix/af_unix.c
index 641f2e4..bc90ddb 100644
--- a/net/unix/af_unix.c
+++ b/net/unix/af_unix.c
@@ -880,10 +880,8 @@ static int unix_bind(struct socket *sock, struct sockaddr *uaddr, int

```
addr_len)
   if (err)
    goto out_mknod_dput;
   err = security_path_mknod(&path, dentry, mode, 0);
-  if (err)
-   goto out_mknod_drop_write;
-  err = vfs_mknod(path.dentry->d_inode, dentry, mode, 0);
-out_mknod_drop_write:
+  if (!err)
+   err = vfs_mknod(path.dentry->d_inode, dentry, mode, 0);
   mnt_drop_write(path.mnt);
   if (err)
    goto out_mknod_dput;
@@ -896,9 +894,9 @@ out_mknod_drop_write:

  spin_lock(&unix_table_lock);

+ err = -EADDRINUSE;
  if (!sun_path[0]) {
-  err = -EADDRINUSE;
-  if (__unix_find_socket_byname(net, sunaddr, addr_len,
+  if (__unix_find_socket_byname(net, sunaddr, addr->len,
            sk->sk_type, hash)) {
    unix_release_addr(addr);
    goto out_unlock;
@@ -906,7 +904,7 @@ out_mknod_drop_write:

   list = &unix_socket_table[addr->hash];
  } else {
-  list = &unix_socket_table[dentry->d_inode->i_ino & (UNIX_HASH_SIZE-1)];
+  list = &unix_socket_table[path.dentry->d_inode->i_ino & (UNIX_HASH_SIZE-1)];
   u->path = path;
  }
```

Subject: [RFC PATCH 2/5] net: split unix_bind()

Posted by Stanislav Kinsbursky on Wed, 15 Aug 2012 16:22:09 GMT

View Forum Message <> Reply to Message

This patch moves UNIX socket insert into separated function, because this code
will be used for unix_fbind() too.

Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>

```
---
 net/unix/af_unix.c |   52 ++++++++++++++++++++++++++++++----------------------
 1 files changed, 29 insertions(+), 23 deletions(-)

diff --git a/net/unix/af_unix.c b/net/unix/af_unix.c
```

```
index bc90ddb..b26200d 100644
--- a/net/unix/af_unix.c
+++ b/net/unix/af_unix.c
@@ -814,11 +814,38 @@ fail:
  return NULL;
 }

+static int __unix_add_sock(struct path *path, struct sock *sk,
+    struct unix_address *addr, int hash)
+{
+ struct net *net = sock_net(sk);
+ struct unix_sock *u = unix_sk(sk);
+ struct sockaddr_un *sunaddr = addr->name;
+ char *sun_path = sunaddr->sun_path;
+ struct hlist_head *list;
+
+ if (!sun_path[0]) {
+  if (__unix_find_socket_byname(net, sunaddr, addr->len,
+          sk->sk_type, hash)) {
+   unix_release_addr(addr);
+   return -EADDRINUSE;
+  }
+
+  list = &unix_socket_table[addr->hash];
+ } else {
+  list = &unix_socket_table[path->dentry->d_inode->i_ino & (UNIX_HASH_SIZE-1)];
+  u->path = *path;
+ }
+
+ __unix_remove_socket(sk);
+ u->addr = addr;
+ __unix_insert_socket(list, sk);
+ return 0;
+
+}

 static int unix_bind(struct socket *sock, struct sockaddr *uaddr, int addr_len)
 {
  struct sock *sk = sock->sk;
- struct net *net = sock_net(sk);
  struct unix_sock *u = unix_sk(sk);
  struct sockaddr_un *sunaddr = (struct sockaddr_un *)uaddr;
  char *sun_path = sunaddr->sun_path;
@@ -827,7 +854,6 @@ static int unix_bind(struct socket *sock, struct sockaddr *uaddr, int
addr_len)
  int err;
  unsigned int hash;
  struct unix_address *addr;
```

```
- struct hlist_head *list;

  err = -EINVAL;
  if (sunaddr->sun_family != AF_UNIX)
@@ -893,27 +919,7 @@ static int unix_bind(struct socket *sock, struct sockaddr *uaddr, int
addr_len)
  }

  spin_lock(&unix_table_lock);
-
- err = -EADDRINUSE;
- if (!sun_path[0]) {
-  if (__unix_find_socket_byname(net, sunaddr, addr->len,
-          sk->sk_type, hash)) {
-   unix_release_addr(addr);
-   goto out_unlock;
-  }
-
-  list = &unix_socket_table[addr->hash];
- } else {
-  list = &unix_socket_table[path.dentry->d_inode->i_ino & (UNIX_HASH_SIZE-1)];
-  u->path = path;
- }
-
- err = 0;
- __unix_remove_socket(sk);
- u->addr = addr;
- __unix_insert_socket(list, sk);
-
-out_unlock:
+ err = __unix_add_sock(&path, sk, addr, hash);
  spin_unlock(&unix_table_lock);
 out_up:
  mutex_unlock(&u->readlock);
```

Subject: [RFC PATCH 3/5] net: new protocol operation fbind() introduced
Posted by Stanislav Kinsbursky on Wed, 15 Aug 2012 16:22:14 GMT
View Forum Message <> Reply to Message

This operation is used to bind socket to specified file.

Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>
---
 include/linux/net.h | 2 ++
 1 files changed, 2 insertions(+), 0 deletions(-)

diff --git a/include/linux/net.h b/include/linux/net.h

```
index e9ac2df..843cb75 100644
--- a/include/linux/net.h
+++ b/include/linux/net.h
@@ -157,6 +157,7 @@ struct kiocb;
 struct sockaddr;
 struct msghdr;
 struct module;
+struct path;

 struct proto_ops {
  int  family;
@@ -165,6 +166,7 @@ struct proto_ops {
  int  (*bind)      (struct socket *sock,
           struct sockaddr *myaddr,
           int sockaddr_len);
+ int  (*fbind)     (struct file *file, struct socket *sock);
  int  (*connect)   (struct socket *sock,
           struct sockaddr *vaddr,
           int sockaddr_len, int flags);
```

Subject: [RFC PATCH 4/5] net: fbind() for unix sockets protocol operations introduced

Posted by Stanislav Kinsbursky on Wed, 15 Aug 2012 16:22:20 GMT

View Forum Message <> Reply to Message

Path for unix_address is taken from passed file.
File inode have to be socket.
Since no sunaddr is present, addr->name is constructed at the place. It
obviously means, then path name can be truncated is it's longer then
UNIX_MAX_PATH.

Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>

```
---
 net/unix/af_unix.c |   78 ++++++++++++++++++++++++++++++++++++++++++++++++++++-
 1 files changed, 76 insertions(+), 2 deletions(-)

diff --git a/net/unix/af_unix.c b/net/unix/af_unix.c
index b26200d..2f34c9d 100644
--- a/net/unix/af_unix.c
+++ b/net/unix/af_unix.c
@@ -286,12 +286,11 @@ static inline struct sock *unix_find_socket_byname(struct net *net,
  return s;
 }

-static struct sock *unix_find_socket_byinode(struct inode *i)
+static struct sock *__unix_find_socket_byinode(struct inode *i)
 {
```

```
  struct sock *s;
  struct hlist_node *node;

- spin_lock(&unix_table_lock);
  sk_for_each(s, node,
       &unix_socket_table[i->i_ino & (UNIX_HASH_SIZE - 1)]) {
   struct dentry *dentry = unix_sk(s)->path.dentry;
@@ -303,6 +302,15 @@ static struct sock *unix_find_socket_byinode(struct inode *i)
  }
  s = NULL;
 found:
+ return s;
+}
+
+static struct sock *unix_find_socket_byinode(struct inode *i)
+{
+ struct sock *s;
+
+ spin_lock(&unix_table_lock);
+ s = __unix_find_socket_byinode(i);
  spin_unlock(&unix_table_lock);
  return s;
 }
@@ -502,6 +510,7 @@ out:

 static int unix_release(struct socket *);
 static int unix_bind(struct socket *, struct sockaddr *, int);
+static int unix_fbind(struct file *file, struct socket *sock);
 static int unix_stream_connect(struct socket *, struct sockaddr *,
           int addr_len, int flags);
 static int unix_socketpair(struct socket *, struct socket *);
@@ -542,6 +551,7 @@ static const struct proto_ops unix_stream_ops = {
  .owner = THIS_MODULE,
  .release = unix_release,
  .bind =  unix_bind,
+ .fbind = unix_fbind,
  .connect = unix_stream_connect,
  .socketpair = unix_socketpair,
  .accept = unix_accept,
@@ -564,6 +574,7 @@ static const struct proto_ops unix_dgram_ops = {
  .owner = THIS_MODULE,
  .release = unix_release,
  .bind =  unix_bind,
+ .fbind = unix_fbind,
  .connect = unix_dgram_connect,
  .socketpair = unix_socketpair,
  .accept = sock_no_accept,
@@ -586,6 +597,7 @@ static const struct proto_ops unix_seqpacket_ops = {
```

```
  .owner = THIS_MODULE,
  .release = unix_release,
  .bind =  unix_bind,
+ .fbind = unix_fbind,
  .connect = unix_stream_connect,
  .socketpair = unix_socketpair,
  .accept = unix_accept,
@@ -843,6 +855,68 @@ static int __unix_add_sock(struct path *path, struct sock *sk,

 }

+static int unix_fbind(struct file *f, struct socket *sock)
+{
+ struct sock *sk = sock->sk, *tmp;
+ struct unix_sock *u = unix_sk(sk);
+ struct unix_address *addr;
+ struct path *path = &f->f_path;
+ struct inode *inode = path->dentry->d_inode;
+ char *buf, *name;
+ int err;
+
+ err = -ENOTSOCK;
+ if (!S_ISSOCK(inode->i_mode))
+  goto out;
+
+ mutex_lock(&u->readlock);
+
+ err = -EINVAL;
+ if (u->addr)
+  goto out_up;
+
+ err = -ENOMEM;
+ buf = (char*)__get_free_page(GFP_KERNEL);
+ if (!buf)
+  goto out_up;
+
+ err = -EFAULT;
+ name = d_path(path, buf, PAGE_SIZE);
+ if (IS_ERR(name))
+  goto out_up_page;
+
+ addr = kmalloc(sizeof(*addr) + sizeof(struct sockaddr_un), GFP_KERNEL);
+ if (!addr)
+  goto out_up_page;
+
+ addr->name->sun_family = AF_UNIX;
+ addr->len = min(strlen(name), (size_t)UNIX_PATH_MAX);
+ memcpy(addr->name->sun_path, name, addr->len);
```

```
+ atomic_set(&addr->refcnt, 1);
+
+ spin_lock(&unix_table_lock);
+
+ err = -EADDRINUSE;
+ tmp = __unix_find_socket_byinode(inode);
+ if (tmp) {
+  sock_put(tmp);
+  unix_release_addr(addr);
+  goto out_unlock;
+ }
+
+ err = __unix_add_sock(path, sk, addr, 0);
+ path_get(path);
+
+out_unlock:
+ spin_unlock(&unix_table_lock);
+out_up_page:
+ free_page((unsigned long)buf);
+out_up:
+ mutex_unlock(&u->readlock);
+out:
+ return err;
+}
+
 static int unix_bind(struct socket *sock, struct sockaddr *uaddr, int addr_len)
 {
  struct sock *sk = sock->sk;
```

Subject: [RFC PATCH 5/5] syscall: sys_fbind() introduced
Posted by Stanislav Kinsbursky on Wed, 15 Aug 2012 16:22:25 GMT
View Forum Message <> Reply to Message

This syscall allows to bind socket to specified file descriptor.
Descriptor can be gained by simple open with O_PATH flag.
Socket node can be created by sys_mknod().

Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>

```
---
 arch/x86/syscalls/syscall_32.tbl |    1 +
 arch/x86/syscalls/syscall_64.tbl |    1 +
 include/linux/syscalls.h         |    1 +
 kernel/sys_ni.c                  |    3 +++
 net/socket.c                     |   25 +++++++++++++++++++++++++
 5 files changed, 31 insertions(+), 0 deletions(-)

diff --git a/arch/x86/syscalls/syscall_32.tbl b/arch/x86/syscalls/syscall_32.tbl
```

```
index 7a35a6e..9594b82 100644
--- a/arch/x86/syscalls/syscall_32.tbl
+++ b/arch/x86/syscalls/syscall_32.tbl
@@ -356,3 +356,4 @@
 347 i386 process_vm_readv sys_process_vm_readv  compat_sys_process_vm_readv
 348 i386 process_vm_writev sys_process_vm_writev  compat_sys_process_vm_writev
 349 i386 kcmp   sys_kcmp
+350 i386 fbind   sys_fbind
diff --git a/arch/x86/syscalls/syscall_64.tbl b/arch/x86/syscalls/syscall_64.tbl
index 51171ae..f964df8 100644
--- a/arch/x86/syscalls/syscall_64.tbl
+++ b/arch/x86/syscalls/syscall_64.tbl
@@ -319,6 +319,7 @@
 310 64 process_vm_readv sys_process_vm_readv
 311 64 process_vm_writev sys_process_vm_writev
 312 64 kcmp   sys_kcmp
+313 common fbind   sys_fbind

 #
 # x32-specific system call numbers start at 512 to avoid cache impact
diff --git a/include/linux/syscalls.h b/include/linux/syscalls.h
index 19439c7..9e78fa4 100644
--- a/include/linux/syscalls.h
+++ b/include/linux/syscalls.h
@@ -602,6 +602,7 @@ asmlinkage long sys_setsockopt(int fd, int level, int optname,
 asmlinkage long sys_getsockopt(int fd, int level, int optname,
     char __user *optval, int __user *optlen);
 asmlinkage long sys_bind(int, struct sockaddr __user *, int);
+asmlinkage long sys_fbind(int, int);
 asmlinkage long sys_connect(int, struct sockaddr __user *, int);
 asmlinkage long sys_accept(int, struct sockaddr __user *, int __user *);
 asmlinkage long sys_accept4(int, struct sockaddr __user *, int __user *, int);
diff --git a/kernel/sys_ni.c b/kernel/sys_ni.c
index dbff751..30c393a 100644
--- a/kernel/sys_ni.c
+++ b/kernel/sys_ni.c
@@ -206,3 +206,6 @@ cond_syscall(compat_sys_open_by_handle_at);

 /* compare kernel pointers */
 cond_syscall(sys_kcmp);
+
+cond_syscall(sys_fbind);
+
diff --git a/net/socket.c b/net/socket.c
index 6e0ccc0..67d9795 100644
--- a/net/socket.c
+++ b/net/socket.c
@@ -1432,6 +1432,31 @@ out:
```

```
 return err;
}

+SYSCALL_DEFINE2(fbind, int, fd, int, sk_fd)
+{
+ struct socket *sock;
+ int err, fput_sk, fput_fd;
+ struct file *file;
+
+ sock = sockfd_lookup_light(sk_fd, &err, &fput_sk);
+ if (!sock)
+  return err;
+
+ err = -EBADF;
+ file = fget_raw_light(fd, &fput_fd);
+ if (!file)
+  goto out_put_sk;
+
+ err = -EINVAL;
+ if (sock->ops->fbind)
+  err = sock->ops->fbind(file, sock);
+
+ fput_light(file, fput_fd);
+out_put_sk:
+ fput_light(sock->file, fput_sk);
+ return err;
+}
+
 /*
  * Bind a name to a socket. Nothing much to do here since it's
  * the protocol's responsibility to handle the local address.
```

Subject: Re: [RFC PATCH 5/5] syscall: sys_fbind() introduced
Posted by hpa on Wed, 15 Aug 2012 16:30:37 GMT
View Forum Message <> Reply to Message

On 08/15/2012 09:22 AM, Stanislav Kinsbursky wrote:
> This syscall allows to bind socket to specified file descriptor.
> Descriptor can be gained by simple open with O_PATH flag.
> Socket node can be created by sys_mknod().
>
> Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>
> ---
> arch/x86/syscalls/syscall_32.tbl | 1 +
> arch/x86/syscalls/syscall_64.tbl | 1 +
> include/linux/syscalls.h | 1 +
> kernel/sys_ni.c | 3 +++

> net/socket.c | 25 ++++++++++++++++++++++++
> 5 files changed, 31 insertions(+), 0 deletions(-)
>
> diff --git a/arch/x86/syscalls/syscall_32.tbl b/arch/x86/syscalls/syscall_32.tbl
> index 7a35a6e..9594b82 100644
> --- a/arch/x86/syscalls/syscall_32.tbl
> +++ b/arch/x86/syscalls/syscall_32.tbl
> @@ -356,3 +356,4 @@
> 347 i386 process_vm_readv sys_process_vm_readv compat_sys_process_vm_readv
> 348 i386 process_vm_writev sys_process_vm_writev compat_sys_process_vm_writev
> 349 i386 kcmp sys_kcmp
> +350 i386 fbind sys_fbind

i386 uses socketcalls... perhaps it shouldn't (socketcalls are pretty
much an abomination), but for socketcall-based architectures this really
should be a socketcall.

Don't you also need fconnect()? Or is that simply handled by allowing
open() without O_PATH?

 -hpa

--
H. Peter Anvin, Intel Open Source Technology Center
I work for Intel. I don't speak on their behalf.

Subject: Re: [RFC PATCH 5/5] syscall: sys_fbind() introduced
Posted by Stanislav Kinsbursky on Wed, 15 Aug 2012 16:43:49 GMT
View Forum Message <> Reply to Message

> On 08/15/2012 09:22 AM, Stanislav Kinsbursky wrote:
>> This syscall allows to bind socket to specified file descriptor.
>> Descriptor can be gained by simple open with O_PATH flag.
>> Socket node can be created by sys_mknod().
>>
>> Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>
>> ---
>> arch/x86/syscalls/syscall_32.tbl | 1 +
>> arch/x86/syscalls/syscall_64.tbl | 1 +
>> include/linux/syscalls.h | 1 +
>> kernel/sys_ni.c | 3 +++
>> net/socket.c | 25 ++++++++++++++++++++++++
>> 5 files changed, 31 insertions(+), 0 deletions(-)
>>
>> diff --git a/arch/x86/syscalls/syscall_32.tbl b/arch/x86/syscalls/syscall_32.tbl
>> index 7a35a6e..9594b82 100644

>> --- a/arch/x86/syscalls/syscall_32.tbl
>> +++ b/arch/x86/syscalls/syscall_32.tbl
>> @@ -356,3 +356,4 @@
>> 347 i386 process_vm_readv sys_process_vm_readv compat_sys_process_vm_readv
>> 348 i386 process_vm_writev sys_process_vm_writev compat_sys_process_vm_writev
>> 349 i386 kcmp sys_kcmp
>> +350 i386 fbind sys_fbind
>
> i386 uses socketcalls... perhaps it shouldn't (socketcalls are pretty
> much an abomination), but for socketcall-based architectures this really
> should be a socketcall.
>

Thanks, Peter. I'll rework this.

> Don't you also need fconnect()? Or is that simply handled by allowing
> open() without O_PATH?
>

Yes, I need it.
If this approach will be accepted, then I'll send one more patch set for fconnect.

> -hpa
>

--
Best regards,
Stanislav Kinsbursky

Subject: Re: [RFC PATCH 0/5] net: socket bind to file descriptor introduced
Posted by Ben Pfaff on Wed, 15 Aug 2012 16:52:53 GMT
View Forum Message <> Reply to Message

Stanislav Kinsbursky <skinsbursky@parallels.com> writes:

> This system call is especially required for UNIX sockets, which has name
> lenght limitation.

The worst of the name length limitations can be worked around by
opening the directory where the socket is to go as a file
descriptor, then using /proc/self/fd/<fd>/<basename> as the name
of the socket. This technique also works with "connect" and in
other contexts where a struct sockaddr is needed. At first
glance, it looks like your patches only help with "bind".

Subject: Re: [RFC PATCH 0/5] net: socket bind to file descriptor introduced
Posted by hpa on Wed, 15 Aug 2012 17:54:15 GMT
View Forum Message <> Reply to Message

On 08/15/2012 09:52 AM, Ben Pfaff wrote:
> Stanislav Kinsbursky <skinsbursky@parallels.com> writes:
>
>> This system call is especially required for UNIX sockets, which has name
>> lenght limitation.
>
> The worst of the name length limitations can be worked around by
> opening the directory where the socket is to go as a file
> descriptor, then using /proc/self/fd/<fd>/<basename> as the name
> of the socket. This technique also works with "connect" and in
> other contexts where a struct sockaddr is needed. At first
> glance, it looks like your patches only help with "bind".
>

The really hard part is what to do with things that are supposed to
return a struct sockaddr. I also have some reservations about using a
new system call to deal with what at least theoretically is only part of
one socket domain.

-hpa

Subject: Re: [RFC PATCH 0/5] net: socket bind to file descriptor introduced
Posted by ebiederm on Wed, 15 Aug 2012 19:49:28 GMT
View Forum Message <> Reply to Message

Stanislav Kinsbursky <skinsbursky@parallels.com> writes:

> This patch set introduces new socket operation and new system call:
> sys_fbind(), which allows to bind socket to opened file.
> File to bind to can be created by sys_mknod(S_IFSOCK) and opened by
> open(O_PATH).
>
> This system call is especially required for UNIX sockets, which has name
> lenght limitation.
>
> The following series implements...

Thinking about this a little more I have serious reservations about this
approach.

Today you are not allowed to bind to an address unless mknod for that
file succeeds. Your patch totally changes those semantics.

Name length limitation does not seeme to justify this at all.

It is possible today to trivially change into a directory and
bind or connect to what would be a long absolute path.

There is also the trick of getting a shorter directory name using
/proc/self/fd if you are threaded and can't change the directory.

The obvious choices at this point are
- Teach bind and connect and af_unix sockets to take longer AF_UNIX
  socket path names.

- introduce sockaddr_fd that can be applied to AF_UNIX sockets,
  and teach unix_bind and unix_connect how to deal with a second type of sockaddr.
  struct sockaddr_fd { short fd_family; short pad; int fd; };

- introduce sockaddr_unix_at that takes a directory file descriptor
  as well as a unix path, and teach unix_bind and unix_connect to deal with a
  second sockaddr type.
  struct sockaddr_unix_at { short family; short pad; int dfd; char path[102]; }
  AF_UNIX_AT

I don't know what the implications of for breaking connect up into 3
system calls and changing the semantics are and I would really rather
not have to think about it.

But it certainly does not look to me like you introduce new systems
calls to do what you want.

Eric

Subject: Re: [RFC PATCH 0/5] net: socket bind to file descriptor introduced
Posted by hpa on Wed, 15 Aug 2012 20:58:36 GMT
View Forum Message <> Reply to Message

On 08/15/2012 12:49 PM, Eric W. Biederman wrote:
>
> There is also the trick of getting a shorter directory name using
> /proc/self/fd if you are threaded and can't change the directory.
>
> The obvious choices at this point are
> - Teach bind and connect and af_unix sockets to take longer AF_UNIX
>   socket path names.
>
> - introduce sockaddr_fd that can be applied to AF_UNIX sockets,
>   and teach unix_bind and unix_connect how to deal with a second type of sockaddr.
>   struct sockaddr_fd { short fd_family; short pad; int fd; };

>
> - introduce sockaddr_unix_at that takes a directory file descriptor
>   as well as a unix path, and teach unix_bind and unix_connect to deal with a
>   second sockaddr type.
>   struct sockaddr_unix_at { short family; short pad; int dfd; char path[102]; }
>   AF_UNIX_AT
>
> I don't know what the implications of for breaking connect up into 3
> system calls and changing the semantics are and I would really rather
> not have to think about it.
>
> But it certainly does not look to me like you introduce new systems
> calls to do what you want.
>

How would you distinguish the new sockaddr types from the traditional
one? New AF_?

 -hpa

Subject: Re: [RFC PATCH 0/5] net: socket bind to file descriptor introduced
Posted by ebiederm on Wed, 15 Aug 2012 21:25:42 GMT
View Forum Message <> Reply to Message

"H. Peter Anvin" <hpa@zytor.com> writes:

> On 08/15/2012 12:49 PM, Eric W. Biederman wrote:
>>
>> There is also the trick of getting a shorter directory name using
>> /proc/self/fd if you are threaded and can't change the directory.
>>
>> The obvious choices at this point are
>> - Teach bind and connect and af_unix sockets to take longer AF_UNIX
>>   socket path names.
>>
>> - introduce sockaddr_fd that can be applied to AF_UNIX sockets,
>>   and teach unix_bind and unix_connect how to deal with a second type of sockaddr.
>>   struct sockaddr_fd { short fd_family; short pad; int fd; };
>>
>> - introduce sockaddr_unix_at that takes a directory file descriptor
>>   as well as a unix path, and teach unix_bind and unix_connect to deal with a
>>   second sockaddr type.
>>   struct sockaddr_unix_at { short family; short pad; int dfd; char path[102]; }
>>   AF_UNIX_AT
>>
>> I don't know what the implications of for breaking connect up into 3
>> system calls and changing the semantics are and I would really rather

>> not have to think about it.
>>
>> But it certainly does not look to me like you introduce new systems
>> calls to do what you want.
>>
>
> How would you distinguish the new sockaddr types from the traditional
> one? New AF_?

Yeah. AF_FD or AF_UNIX_AT is what I was thinking. The way the code
falls out that should be compartively simple to implement.

recvmsg etc would give you sockaddr_un sockets when they come from the
kernel.

Eric

Subject: Re: [RFC PATCH 0/5] net: socket bind to file descriptor introduced
Posted by ebiederm on Thu, 16 Aug 2012 03:03:24 GMT
View Forum Message <> Reply to Message

Stanislav Kinsbursky <skinsbursky@parallels.com> writes:

> This patch set introduces new socket operation and new system call:
> sys_fbind(), which allows to bind socket to opened file.
> File to bind to can be created by sys_mknod(S_IFSOCK) and opened by
> open(O_PATH).
>
> This system call is especially required for UNIX sockets, which has name
> lenght limitation.
>
> The following series implements...

Hmm. I just realized this patchset is even sillier than I thought.

Stanislav is the problem you are ultimately trying to solve nfs clients
in a container connecting to the wrong user space rpciod?

Aka net/sunrpc/xprtsock.c:xs_setup_local only taking an absolute path
and then creating a delayed work item to actually open the unix domain
socket?

The straight correct and straight forward thing to do appears to be:
- Capture the root from current->fs in xs_setup_local.
- In xs_local_finish_connect change current->fs.root to the captured
  version of root before kernel_connect, and restore current->fs.root
  after kernel_connect.

It might not be a bad idea to implement open on unix domain sockets in
a filesystem as create(AF_LOCAL)+connect() which would allow you to
replace __sock_create + kernel_connect with a simple file_open_root.

But I think the simple scheme of:
struct path old_root;
old_root = current->fs.root;
kernel_connect(...);
current->fs.root = old_root;

Is more than sufficient and will remove the need for anything
except a purely local change to get nfs clients to connect from
containers.

Eric

Subject: Re: [RFC PATCH 0/5] net: socket bind to file descriptor introduced
Posted by bfields on Thu, 16 Aug 2012 13:54:29 GMT
View Forum Message <> Reply to Message

On Wed, Aug 15, 2012 at 08:03:24PM -0700, Eric W. Biederman wrote:
> Stanislav Kinsbursky <skinsbursky@parallels.com> writes:
>
> > This patch set introduces new socket operation and new system call:
> > sys_fbind(), which allows to bind socket to opened file.
> > File to bind to can be created by sys_mknod(S_IFSOCK) and opened by
> > open(O_PATH).
> >
> > This system call is especially required for UNIX sockets, which has name
> > lenght limitation.
> >
> > The following series implements...
>
> Hmm. I just realized this patchset is even sillier than I thought.
>
> Stanislav is the problem you are ultimately trying to solve nfs clients
> in a container connecting to the wrong user space rpciod?
>
> Aka net/sunrpc/xprtsock.c:xs_setup_local only taking an absolute path
> and then creating a delayed work item to actually open the unix domain
> socket?
>
> The straight correct and straight forward thing to do appears to be:
> - Capture the root from current->fs in xs_setup_local.
> - In xs_local_finish_connect change current->fs.root to the captured
>   version of root before kernel_connect, and restore current->fs.root

> after kernel_connect.

Ah, yep, that should do it.

--b.

>
> It might not be a bad idea to implement open on unix domain sockets in
> a filesystem as create(AF_LOCAL)+connect() which would allow you to
> replace __sock_create + kernel_connect with a simple file_open_root.
>
> But I think the simple scheme of:
> struct path old_root;
> old_root = current->fs.root;
> kernel_connect(...);
> current->fs.root = old_root;
>
> Is more than sufficient and will remove the need for anything
> except a purely local change to get nfs clients to connect from
> containers.
>
> Eric

Subject: Re: [RFC PATCH 0/5] net: socket bind to file descriptor introduced
Posted by Stanislav Kinsbursky on Mon, 20 Aug 2012 10:18:13 GMT
View Forum Message <> Reply to Message

> Stanislav Kinsbursky <skinsbursky@parallels.com> writes:
>
>> This patch set introduces new socket operation and new system call:
>> sys_fbind(), which allows to bind socket to opened file.
>> File to bind to can be created by sys_mknod(S_IFSOCK) and opened by
>> open(O_PATH).
>>
>> This system call is especially required for UNIX sockets, which has name
>> lenght limitation.
>>
>> The following series implements...
>
> Hmm. I just realized this patchset is even sillier than I thought.
>
> Stanislav is the problem you are ultimately trying to solve nfs clients
> in a container connecting to the wrong user space rpciod?
>

Hi, Eric.

The problem you mentioned was the reason why I started to think about this.
But currently I believe, that limitations in unix sockets connect or bind should
be removed, because it will be useful it least for CRIU project.

> Aka net/sunrpc/xprtsock.c:xs_setup_local only taking an absolute path
> and then creating a delayed work item to actually open the unix domain
> socket?
>
> The straight correct and straight forward thing to do appears to be:
> - Capture the root from current->fs in xs_setup_local.
> - In xs_local_finish_connect change current->fs.root to the captured
>   version of root before kernel_connect, and restore current->fs.root
>   after kernel_connect.
>
> It might not be a bad idea to implement open on unix domain sockets in
> a filesystem as create(AF_LOCAL)+connect() which would allow you to
> replace __sock_create + kernel_connect with a simple file_open_root.
>

I like the idea of introducing new family (AF_LOCAL_AT for example) and new
sockaddr for connecting or binding from specified root. The only thing I'm
worrying is passing file descriptor to unix bind or connect routine. Because
this approach doesn't provide easy way to use such family and sockaddr in kernel
(like in NFS example).

> But I think the simple scheme of:
> struct path old_root;
> old_root = current->fs.root;
> kernel_connect(...);
> current->fs.root = old_root;
>
> Is more than sufficient and will remove the need for anything
> except a purely local change to get nfs clients to connect from
> containers.
>

That was my first idea. And probably it would be worth to change all fs_struct
to support sockets with relative path.
What do you think about it?

> Eric
>

--
Best regards,
Stanislav Kinsbursky

Subject: Re: [RFC PATCH 0/5] net: socket bind to file descriptor introduced
Posted by bfields on Tue, 04 Sep 2012 19:00:07 GMT
View Forum Message <> Reply to Message

On Mon, Aug 20, 2012 at 02:18:13PM +0400, Stanislav Kinsbursky wrote:

> >Stanislav Kinsbursky <skinsbursky@parallels.com> writes:
> >
> >>This patch set introduces new socket operation and new system call:
> >>sys_fbind(), which allows to bind socket to opened file.
> >>File to bind to can be created by sys_mknod(S_IFSOCK) and opened by
> >>open(O_PATH).
> >>
> >>This system call is especially required for UNIX sockets, which has name
> >>lenght limitation.
> >>
> >>The following series implements...
> >
> >Hmm. I just realized this patchset is even sillier than I thought.
> >
> >Stanislav is the problem you are ultimately trying to solve nfs clients
> >in a container connecting to the wrong user space rpciod?
> >
>
> Hi, Eric.
> The problem you mentioned was the reason why I started to think about this.
> But currently I believe, that limitations in unix sockets connect or
> bind should be removed, because it will be useful it least for CRIU
> project.
>
> >Aka net/sunrpc/xprtsock.c:xs_setup_local only taking an absolute path
> >and then creating a delayed work item to actually open the unix domain
> >socket?
> >
> >The straight correct and straight forward thing to do appears to be:
> >- Capture the root from current->fs in xs_setup_local.
> >- In xs_local_finish_connect change current->fs.root to the captured
> > version of root before kernel_connect, and restore current->fs.root
> > after kernel_connect.
> >
> >It might not be a bad idea to implement open on unix domain sockets in
> >a filesystem as create(AF_LOCAL)+connect() which would allow you to
> >replace __sock_create + kernel_connect with a simple file_open_root.
> >
>
> I like the idea of introducing new family (AF_LOCAL_AT for example)
> and new sockaddr for connecting or binding from specified root. The
> only thing I'm worrying is passing file descriptor to unix bind or
> connect routine. Because this approach doesn't provide easy way to

> use such family and sockaddr in kernel (like in NFS example).
>
> >But I think the simple scheme of:
> >struct path old_root;
> >old_root = current->fs.root;
> >kernel_connect(...);
> >current->fs.root = old_root;
> >
> >Is more than sufficient and will remove the need for anything
> >except a purely local change to get nfs clients to connect from
> >containers.
> >
>
> That was my first idea.

So is this what you're planning on doing now?

> And probably it would be worth to change all
> fs_struct to support sockets with relative path.
> What do you think about it?

I didn't understand the question. Are you suggesting that changes to
fs_struct would be required to make this work? I don't see why.

--b.

Subject: Re: [RFC PATCH 0/5] net: socket bind to file descriptor introduced
Posted by bfields on Fri, 05 Oct 2012 20:00:09 GMT
View Forum Message <> Reply to Message

On Tue, Sep 04, 2012 at 03:00:07PM -0400, bfields wrote:
> On Mon, Aug 20, 2012 at 02:18:13PM +0400, Stanislav Kinsbursky wrote:

> > >Stanislav Kinsbursky <skinsbursky@parallels.com> writes:
> > >
> > >>This patch set introduces new socket operation and new system call:
> > >>sys_fbind(), which allows to bind socket to opened file.
> > >>File to bind to can be created by sys_mknod(S_IFSOCK) and opened by
> > >>open(O_PATH).
> > >>
> > >>This system call is especially required for UNIX sockets, which has name
> > >>lenght limitation.
> > >>
> > >>The following series implements...
> > >
> > >Hmm. I just realized this patchset is even sillier than I thought.
> > >

> > >Stanislav is the problem you are ultimately trying to solve nfs clients
> > >in a container connecting to the wrong user space rpciod?
> > >
> >
> > Hi, Eric.
> > The problem you mentioned was the reason why I started to think about this.
> > But currently I believe, that limitations in unix sockets connect or
> > bind should be removed, because it will be useful it least for CRIU
> > project.
> >
> > >Aka net/sunrpc/xprtsock.c:xs_setup_local only taking an absolute path
> > >and then creating a delayed work item to actually open the unix domain
> > >socket?
> > >
> > >The straight correct and straight forward thing to do appears to be:
> > >- Capture the root from current->fs in xs_setup_local.
> > >- In xs_local_finish_connect change current->fs.root to the captured
> > > version of root before kernel_connect, and restore current->fs.root
> > > after kernel_connect.
> > >
> > >It might not be a bad idea to implement open on unix domain sockets in
> > >a filesystem as create(AF_LOCAL)+connect() which would allow you to
> > >replace __sock_create + kernel_connect with a simple file_open_root.
> > >
> >
> > I like the idea of introducing new family (AF_LOCAL_AT for example)
> > and new sockaddr for connecting or binding from specified root. The
> > only thing I'm worrying is passing file descriptor to unix bind or
> > connect routine. Because this approach doesn't provide easy way to
> > use such family and sockaddr in kernel (like in NFS example).
> >
> > >But I think the simple scheme of:
> > >struct path old_root;
> > >old_root = current->fs.root;
> > >kernel_connect(...);
> > >current->fs.root = old_root;
> > >
> > >Is more than sufficient and will remove the need for anything
> > >except a purely local change to get nfs clients to connect from
> > >containers.
> > >
> >
> > That was my first idea.
>
> So is this what you're planning on doing now?

What ever happened to this?

--b.

>
> > And probably it would be worth to change all
> > fs_struct to support sockets with relative path.
> > What do you think about it?
>
> I didn't understand the question. Are you suggesting that changes to
> fs_struct would be required to make this work? I don't see why.
>
> --b.

Subject: Re: [RFC PATCH 0/5] net: socket bind to file descriptor introduced
Posted by Stanislav Kinsbursky on Mon, 08 Oct 2012 08:37:51 GMT
View Forum Message <> Reply to Message

> On Tue, Sep 04, 2012 at 03:00:07PM -0400, bfields wrote:
>> On Mon, Aug 20, 2012 at 02:18:13PM +0400, Stanislav Kinsbursky wrote:

>>>> Stanislav Kinsbursky <skinsbursky@parallels.com> writes:
>>>>
>>>>> This patch set introduces new socket operation and new system call:
>>>>> sys_fbind(), which allows to bind socket to opened file.
>>>>> File to bind to can be created by sys_mknod(S_IFSOCK) and opened by
>>>>> open(O_PATH).
>>>>>
>>>>> This system call is especially required for UNIX sockets, which has name
>>>>> lenght limitation.
>>>>>
>>>>> The following series implements...
>>>>
>>>> Hmm. I just realized this patchset is even sillier than I thought.
>>>>
>>>> Stanislav is the problem you are ultimately trying to solve nfs clients
>>>> in a container connecting to the wrong user space rpciod?
>>>>
>>>
>>> Hi, Eric.
>>> The problem you mentioned was the reason why I started to think about this.
>>> But currently I believe, that limitations in unix sockets connect or
>>> bind should be removed, because it will be useful it least for CRIU
>>> project.
>>>
>>>> Aka net/sunrpc/xprtsock.c:xs_setup_local only taking an absolute path
>>>> and then creating a delayed work item to actually open the unix domain
>>>> socket?

>>>>
>>>> The straight correct and straight forward thing to do appears to be:
>>>> - Capture the root from current->fs in xs_setup_local.
>>>> - In xs_local_finish_connect change current->fs.root to the captured
>>>>   version of root before kernel_connect, and restore current->fs.root
>>>>   after kernel_connect.
>>>>
>>>> It might not be a bad idea to implement open on unix domain sockets in
>>>> a filesystem as create(AF_LOCAL)+connect() which would allow you to
>>>> replace __sock_create + kernel_connect with a simple file_open_root.
>>>>
>>>
>>> I like the idea of introducing new family (AF_LOCAL_AT for example)
>>> and new sockaddr for connecting or binding from specified root. The
>>> only thing I'm worrying is passing file descriptor to unix bind or
>>> connect routine. Because this approach doesn't provide easy way to
>>> use such family and sockaddr in kernel (like in NFS example).
>>>
>>>> But I think the simple scheme of:
>>>> struct path old_root;
>>>> old_root = current->fs.root;
>>>> kernel_connect(...);
>>>> current->fs.root = old_root;
>>>>
>>>> Is more than sufficient and will remove the need for anything
>>>> except a purely local change to get nfs clients to connect from
>>>> containers.
>>>>
>>>
>>> That was my first idea.
>>
>> So is this what you're planning on doing now?
>
> What ever happened to this?
>

Sorry, was busy.
I'll prepare patch today, I hope.

> --b.
>
>>
>>> And probably it would be worth to change all
>>> fs_struct to support sockets with relative path.
>>> What do you think about it?
>>
>> I didn't understand the question. Are you suggesting that changes to
>> fs_struct would be required to make this work? I don't see why.

>>
>> --b.

--
Best regards,
Stanislav Kinsbursky