Sparse

Sparse is a semantic checker for C programs; it can be used to find a number of potential problems with kernel code. See https://lwn.net/Articles/689907/ for an overview of sparse; this document contains some kernel-specific sparse information. More information on sparse, mainly about its internals, can be found in its official pages at https://sparse.docs.kernel.org.

Using sparse for typechecking

"__bitwise" is a type attribute, so you have to do something like this:

```
typedef int __bitwise pm_request_t;

enum pm_request {
        PM_SUSPEND = (__force pm_request_t) 1,
        PM_RESUME = (__force pm_request_t) 2
};
```

which makes PM_SUSPEND and PM_RESUME "bitwise" integers (the "__force" is there because sparse will complain about casting to/from a bitwise type, but in this case we really _do_ want to force the conversion). And because the enum values are all the same type, now "enum pm_request" will be that type too.

And with gcc, all the "__bitwise"/"__force stuff" goes away, and it all ends up looking just like integers to gcc.

Quite frankly, you don't need the enum there. The above all really just boils down to one special "int __bitwise" type.

So the simpler way is to just do:

```
typedef int __bitwise pm_request_t;

#define PM_SUSPEND ((__force pm_request_t) 1)
#define PM_RESUME ((__force pm_request_t) 2)
```

and you now have all the infrastructure needed for strict typechecking.

One small note: the constant integer "0" is special. You can use a constant zero as a bitwise integer type without sparse ever complaining. This is because "bitwise" (as the name implies) was designed for making sure that bitwise types don't get mixed up (little-endian vs big-endian vs cpu-endian vs whatever), and there the constant "0" really _is_ special.

Using sparse for lock checking

The following macros are undefined for gcc and defined during a sparse run to use the "context" tracking feature of sparse, applied to locking. These annotations tell sparse when a lock is held, with regard to the annotated function's entry and exit.

__must_hold - The specified lock is held on function entry and exit.

__acquires - The specified lock is held on function exit, but not entry.

__releases - The specified lock is held on function entry, but not exit.

If the function enters and exits without the lock held, acquiring and releasing the lock inside the function in a balanced way, no annotation is needed. The three annotations above are for cases where sparse would otherwise report a context imbalance.

Getting sparse

You can get tarballs of the latest released versions from: https://www.kernel.org/pub/software/devel/sparse/dist/

Alternatively, you can get snapshots of the latest development version of sparse using git to clone:

```
git://git.kernel.org/pub/scm/devel/sparse/sparse.git
```

Once you have it, just do:

```
make
make install
```

as a regular user, and it will install sparse in your ~/bin directory.

Using sparse

Do a kernel make with "make C=1" to run sparse on all the C files that get recompiled, or use "make C=2" to run sparse on the files whether they need to be recompiled or not. The latter is a fast way to check the whole tree if you have already built it.

The optional make variable CF can be used to pass arguments to sparse. The build system passes -Wbitwise to sparse automatically.

Note that sparse defines the __CHECKER__ preprocessor symbol.