DMS ClientNode Coding Style

The coding style of DMS ClientNode tries to provide following features for development and maintenance:

1. Easier for human reading and understanding.
2. Provide better run time performance and compiler performance.
3. Prevent some errors that hard to debug.

Chapter 1: file header of .c files and .h files

Each files of DMS ClientNode including following words at the beginning of file:

/\*

\* Copyright (C) 2010-2013 by Cloud Computing Center for Mobile Applications

\* Industrial Technology Research Institute

\*

\* dms\_client\_mm.c

\*/

Copyright announcement and file name.

NOTE: Darren will release an official banner.

Chapter 2: File types of a component:

Basically, a .c file including a components. A component X can be across several .c files, but here we treat that each .c is a sub-component of X. Each .c file contain two .h files, one is private file which including things used inside this component, the other is public file which including things that can be accessed by other components. The previous one named as XXXXX\_private.h and later one named as XXXX.h.

The function placed at XXXX\_private.h will add a modifier “static”.

The function and variable placed at XXXX.h will add a modifier “extern”

Chapter 4: Placement of functions inside a .c file

There are many functions placed a .c file, the basic rule for C language is

Rule 1: Place caller function after called function. So that other programmer can read from bottom to top.

Rule 2: Place functions that share similar abstract concept close to each other.

For example:

int enqueue\_item\_to\_queue(item\_t item) {

….

}

item\_t dequeue\_item\_from\_queue(item\_t item) {

….

}

void DO\_A(void) {

….

}

void DO\_B(void) {

….

}

Avoid:

int enqueue\_item\_to\_queue(item\_t item) {

….

}

void DO\_A(void) {

….

}

void DO\_B(void) {

….

}

item\_t dequeue\_item\_from\_queue(item\_t item) {

….

}

For example:

int enqueue\_item\_to\_queue(item\_t item) {

….

}

static item\_t dequeue\_item\_from\_queue(item\_t item) {

….

}

static void DO\_A(void) {

….

}

static void DO\_B(void) {

….

}

Avoid:

item\_t dequeue\_item\_from\_queue(item\_t item) {

….

}

static void DO\_A(void) {

….

}

static void DO\_B(void) {

….

}

int enqueue\_item\_to\_queue(item\_t item) {

….

}

Rule 3: Place static function close to each other.

Chapter 4: include .h file in header file

Rule 1: Try to avoid include .h file in header file. This will degrade the compile performance. Try to use forward declaration.

Rule 2: If you want to provide a library and you don’t want user include many header files, you can provide a single header file which including all .h files in it. And user who using this library will include this header only.

Chapter 4: Indentation

DMS ClientNode using 4 spaces characters for indent.

switch statement indent [2]:

switch (suffix) {

case 'G':

case 'g':

mem <<= 30;

break;

case 'M':

case 'm':

mem <<= 20;

break;

default:

break;

}

Don't put multiple statements on a single line unless you have something to hide [2].

Don't put multiple assignments on a single line either. [2]

Chapter 5: Breaking long lines and strings [2]

The limit on the length of lines is 80 columns and this is a strongly preferred limit.

Chapter 6: Placing Braces and Spaces [2]:

Chapter 6-1: Braces

Put the opening brace last on the line, and put the closing brace first, thusly:

if (x is true) {

we do y

}

This applies to all non-function statement blocks (if, switch, for, while, do).

switch (action) {

case KOBJ\_ADD:

return "add";

case KOBJ\_REMOVE:

return "remove";

default:

return NULL;

}

Brace about function: opening brace at the beginning of the next line

int function (int x)

{

body of function

}

Note that the closing brace is empty on a line of its own, \_except\_ in the cases where it is followed by a continuation of the same statement, ie a "while" in a do-statement or an "else" in an if-statement, like this:

do {

body of do-loop

} while (condition);

and

if (x == y) {

….

} else if (x > y) {

….

} else {

….

}

Don’t write as following:

if (condition)

action();

and

if (condition)

action();

else

do\_that();

Chapter 6-2: spaces [2]

So use a space after these keywords:

if, switch, case, for, do, while

but not with sizeof, typeof, alignof, or \_\_attribute\_\_. E.g.,

s = sizeof(struct file);

Do not add spaces around (inside) parenthesized expressions. This example is \*bad\*:

s = sizeof( struct file );

When declaring pointer data or a function that returns a pointer type, the preferred use of '\*' is adjacent to the data name or function name and not adjacent to the type name. Examples:

char \*linux\_banner;

unsigned long long memparse(char \*ptr, char \*\*retptr);

char \*match\_strdup(substring\_t \*s);

Use one space around (on each side of) most binary and ternary operators, such as any of these:

= + - < > \* / % | & ^ <= >= == != ? :

but no space after unary operators:

& \* + - ~ ! sizeof typeof alignof \_\_attribute\_\_ defined

no space before the postfix increment & decrement unary operators:

++ --

no space after the prefix increment & decrement unary operators:

++ --

and no space around the '.' and "->" structure member operators.

Chapter 7: Naming

As simple as possible.

Chapter 8: Typedefs

Naming : struct XXX -> XXX\_t

Chapter 9: Functions

Function return value: bool or integer

If your function is check whether a feature is enable or disable, using bool:

For example:

bool is\_metadata\_cache\_enable (volume\_device\_t \*vol)

if (vol->metadata\_enable) {

return true;

} else {

return false;

}

if(is\_metadata\_cache\_enable(vol)) {

add\_to\_cache();

}

If your function is performs some action and return value indicate that the result of action is success or fail, using integer. And return 0 when success, return -1 when fail.

This is because caller will perform checking as following:

int32\_t do\_io(io\_request\_t \*ior)

{

if(ior == NULL) {

return -1;

}

rw\_dick(ior);

return 0;

}

if(do\_io(ior)) {

printk(“do io error\n”);

return;

}

Chapter 10: Centralized exiting of functions

Avoid multiple returns from the same fuction

Chapter 11: goto lable:

Using uppercase for all characters of a goto label, this can reminder programmer that here has a goto. Put lable at left most.

if (a is true) {

goto FUNCTION\_OUT;

}

return true;

FUNCTION\_OUT:

return false;

Chapter 12: Commenting

Chapter 13: Data structures [2]

Data structures that have visibility outside the single-threaded environment they are created and destroyed in should always have reference counts.

Chapter 14: Macros, Enums and RTL [2]

Uppercase letter for Macros.

Chapter 15: Function return values and names

Chapter 16: assertion and WARN\_ON(true)

There is no assertion at kernel, but you can using WARN\_ON(true) to indicate that an error happens.

For example:

switch (opcode) {

case A:

return A;

break;

case B:

return B;

break;

default:

WARN\_ON(true);

return C;

}

An opcode either A or B, other is impossible. We place a WARN\_ON(true) to indicate something wrong.

Chapter 17: Declaration and initialization

Avoid mix declaration and initialization. Initial a variable only when you are try to use it.

Example:

void Funt\_A (int \*a)

{

int ret;

if(a is NULL) {

return;

}

ret = 1;

\*a = \*a + ret;

return;

}

Avoid

void Funt\_A (int \*a)

{

int ret = 1;

if(a is NULL) {

return;

}

\*a = \*a + ret;

return;

}

Chapter 18: Assignment and Comparison – A suggestion

A frequently errors of coding is write comparison as assignment. And it is hard to debug. Here we suggest that if programmer want to compare a variable with a constant, per-define variable, or a number. Using format:

if (constant == A) {

do\_action();

}

Avoid using

if(A == constant) {

do\_action();

}

Because the compiler won’t know the difference between previous one with following:

if(A = constant) {

do\_action();

}

Reference:

[1] GNU standard <http://www.gnu.org/prep/standards/standards.html>

[2] Linux kernel standard <http://lxr.linux.no/linux/Documentation/CodingStyle>

[3] Google C++ standard <http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml>