

Introduction to Extended Common Coupling with an Application Study on Linux

Liguo Yu
Computer Science and Informatics
Indiana University South Bend
1700 Mishawaka Ave. P.O. Box 7111
South Bend, IN 46634, USA
ligyu@iusb.edu

Srini Ramaswamy
Computer Science Department
University of Arkansas at Little Rock
2801 S. University Avenue
Little Rock, AR 72204, USA
sxramaswamy@ualr.edu

ABSTRACT

Common coupling is usually considered an obstacle to software maintenance. In this paper, we study the interaction of common coupling with other forms of software coupling and introduce the concept of extended common coupling. We define four types of extended common coupling, which can help us to understand how a global variable is usually referenced, and how it is related to other forms of coupling. We perform an application study on Linux global variable `current`. The result shows that in most cases, global variable `current` in Linux is used with stamp coupling instead of data coupling, which will make the maintenance more difficult.

Categories and Subject Descriptors

D.2.2.d [Modules and interfaces], D.2.7.g [Maintainability], D.2.8 [Metrics/ Measurement], D.2.10.h [Quality analysis and evaluations].

General Terms

Design.

Keywords

Modularity, common coupling, maintenance.

1. INTRODUCTION

A module is an independent piece of software that has its own name space and identifier. Coupling is defined in the context of structured design as “the measure of the strength of association established by a connection from one module to another” [1]. Many different forms of coupling have been identified, including data coupling, stamp coupling, control coupling, common coupling, and content coupling [2]. Table 1 lists the definitions of these various forms of coupling [3].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACM SE'06, March 10–12, 2006, Melbourne, Florida, USA.
Copyright 2006 ACM 1-59593-315-8/06/04...\$5.00.

A good software system should have high cohesion within each module and low coupling between modules. Strong coupling means a high degree of dependency between software modules. The strength of coupling in Table 1 increases from top to bottom. This means data coupling is better than stamp coupling, control coupling is better than common coupling.

Table 1. Definitions of various forms of coupling [3]

Name	Definition
Data coupling	Two modules are data coupled if they pass data through a parameter that is a simple data type or a data structure in which all elements are used by the called module.
Stamp coupling	Two modules are stamp coupled if they pass data through a parameter that is a data structure, but the called module operates on only some of the individual fields of that data structure.
Control coupling	Two modules are control coupled if one passes a variable to the other that is used to control the internal logic of the other.
Common coupling	Two modules are common coupled if they refer to the same global variable.
Content coupling	Two modules are content coupled if one module refers to the inside content of the other in any way.

It should be noted here, a data structure could induce either data coupling or stamp coupling [4]. For example, in the function `extract_taxpayer_info (taxpayer_record)`, two modules interact via a data structure that contains all the information on a taxpayer, and called function manipulates all fields of that data structure, so they are considered data-coupled via `taxpayer_record`. Consider another function `calculate_taxrefund (taxpayer_record)`, which passes the same data structure, but the called function uses only some of these data fields, such as `total_income` and `tax_paid`. In this case, `taxpayer_record` induces stamp coupling between modules.

Two modules are content-coupled if one module refers to the inside content of the other in any way. For example, one module branches or falls through into another, or one module refers to data within another. Content coupling defies the principle of modularity and should never be used. Modern programming

language such as C, C++, and Java prevents the use of content coupling by their predefined language rules.

In contrast, common coupling is a strong form of coupling and it is supported by most programming languages such as C, C++. Common coupling induces strong dependencies between software modules, making software modules difficult to understand, maintain, and reuse [5]. Therefore, a lot of studies are performed on common coupling [5] [6].

Common coupling is related with the definition and use of a global variable [6]. Each occurrence of a variable in source code is either a definition of that variable or a use of that variable. A *definition* of a variable x is a statement that assigns a value to x . The most common form of definition is an assignment statement, such as $x = 12$. The *use* of a variable x is a statement that utilizes the value of x , such as $y = x - 5$. From the viewpoint of maintenance, changes to a definition of global variable can affect the use of that global variable. Therefore, it needs more effort to make change to a definition of an instance of a global variable.

Two modules may be coupled in more than one way. Conventionally, we consider only the worst level of coupling present. For example, if two modules are stamp- and common-coupled, we consider them to be common-coupled. Thus, if there is common coupling, we ignore all better forms of coupling. For example, suppose gv is a global variable, which is declared as an integer, and x is a local variable also declared as an integer. The variable gv in the statement $x = gv$ induces common coupling between this module and the other module that accesses gv . However, because gv is a scalar and its value is passed between modules, it also induces data coupling between these two modules. But, in practice, we ignore the data coupling between these two modules and consider them as only common-coupled via gv .

The reason for this approach is that it cannot happen that only common coupling (and no other forms of coupling) is present. Every instance of a global variable is either a definition or a use, and thus there will always be control, stamp, or data coupling as well. Also, wherever there is control coupling, there has to be stamp or data coupling, because the value of the control variable is used. In order to avoid possible confusion by listing multiple coupling forms, it is usual to ignore the better levels of coupling and to concentrate on the potentially most dangerous level of coupling present.

In this study, we consider all the levels of coupling a global variable induces between modules. For the above example, we consider the coupling between these two modules as both common coupling and data coupling, and call it “data-common coupling.” Because we are interested in only the coupling induced by global variables, we call this kind of classification of common coupling *extended common coupling*. The above “data-common coupling” is one kind of extended common coupling.

The purpose of this study is to understand how a global variable is usually referenced, and how it is related to other forms of coupling, such as data coupling, stamp coupling, and control coupling. The remainder of this paper is organized as follows: Section 2 defines different types of extended common coupling. Section 3 is our application study on Linux. Section 4 contains our conclusions.

2. DEFINITIONS OF EXTENDED COMMON COUPLING

First we define four types of extended common coupling, as shown in Table 2. The detailed explanation is given below.

Table 2. The definitions of extended common coupling

Type	Name	Definition
A	stamp-common coupling	Two modules are stamp-common coupled if an instance of a global variable induces both common coupling and stamp coupling between the two modules.
B	data-common coupling	Two modules are data-common coupled if an instance of a global variable induces both common coupling and data coupling between the two modules.
C	stamp-control-common coupling	Two modules are stamp-control-common coupled if an instance of a global variable induces common coupling, stamp coupling, and control coupling between the two modules.
D	data-control-common coupling	Two modules are data-control-common coupled if an instance of a global variable induces common coupling, data coupling, and control coupling between the two modules.

- Type A – stamp-common coupling. Stamp-common coupling occurs between two modules when an instance of a global variable induces both common coupling and stamp coupling between these two modules. For example, suppose module A and module B both access global variable `process`, a pointer to a data structure. In the following statement of module A,

```
process->state = TASK_RUNNING;
```

the instance of `process` induces common coupling between module A and module B. Only the field `state` of `process` is referenced, accordingly, this instance of `process` also induces stamp coupling between module A and module B, because the value of data structure `process` is passed between modules, and module A operates on only some fields of this data structure. We call this kind of extended common coupling “stamp-common coupling.”

- Type B – data-common coupling. Data-common coupling occurs between two modules when an instance of a global variable induces both common coupling and data coupling between these two modules. For example, suppose module A and module B both access global variable `process`, a composite data type. In the following statement of module A,

```
return process;
```

the instance of `process` induces common coupling between module A and module B. Because data structure `process` is passed between modules and module A operates on all the fields of `process`, this instance of `process` also induces data coupling between module A and module B. We call this type of extended common coupling “data-common coupling.”

- Type C – stamp-control-common coupling. Two modules are considered stamp-control-common coupled if an instance of a global variable induces common coupling, stamp coupling, and control coupling between these two modules. For example, suppose module A and module B both access global variable `process`. In the following statement of module A, `x` is a local variable.

```
if (process->need_resched) x=1;
else x=0;
```

As explained in Type A, `process` induces both common coupling and stamp coupling between module A and module B. Also, `process` is a global variable referenced by both modules, and module B can control the internal logic of module A by changing the value of `process`. So `process` also induces control coupling between module A and module B. This is a Type C extended common coupling. It can be interpreted as “two modules access the same global variable and some fields of a global variable are used to control the internal logic of one module.”

- Type D – data-control-common coupling. Two modules are considered data-control-common coupled if an instance of a global variable induces common coupling, data coupling, and control coupling between these two modules. For example, suppose module A and module B both access global variable `process`. In the following statement of module A,

```
if (p!= process) p->pid=1;
```

the instance of `process` induces both common coupling and data coupling between module A and module B. Also, module B can control the internal logic of module A by changing the value of `process`, so module A and module B are control-coupled via `process`. Therefore, the instance of `process` also induces control coupling between module A and module B. This is a Type D extended common coupling, which can be interpreted as “two modules access the same global variable and the whole fields of a global variable are used to control the internal logic of one module.” it is either assigned a new value or its present value is used.

3. APPLICATION STUDY ON LINUX

Global variable `current` was first introduced in version 1.0.0 of Linux. In version 1.0.9, it was defined as a pointer to a structure `task_struct` in kernel module `sched.c`:

```
struct task_struct *current = &init_task;
```

From version 1.3 onward, `current` was defined as a preprocessor macro `get_current ()`, which is an inline function that returns a pointer to a structure `task_struct`. In both cases, `current` can be viewed as a pointer to a structure `task_struct`, which describes a task in the system. During scheduling, the kernel relies upon a linked list of tasks to determine which task should be run next. This linked list is a list of data structures of type `task_struct` [7].

Using LXR (Linux Cross Reference) tool, we studied global variable `current` in Linux version 2.4.20. First, we examined all the instances of `current` in both kernel module and nonkernel modules and performed a definition-use analysis. The result is shown in Figure 1 [6]. Second, we determined the type of extended common coupling induced by each instance of `current` in kernel modules. Third, different types of extended common coupling were summarized and analyzed.

Table 3 shows the number of instances of definitions and uses of `current` in Linux 2.4.20. It shows that `current` appears in 18 kernel modules and 1071 nonkernel modules. There are 114 instances of definitions and 382 instances of uses in kernel modules and 1403 instances of definitions and 6795 instances of uses in nonkernel modules. To make our experiment manageable, we studied in detail only the 496 instances of `current` in kernel modules.

Table 3. Number of instances of definitions and uses of `current` in Linux 2.4.20

	Number of modules	Number of definitions	Number of uses
kernel	18	114	382
nonkernel	1071	1403	6795

By examining the 496 instances of `current` in kernel modules, we found each of these instances induces one of the four types of extended common coupling described above. The distribution of these four types of extended common coupling is listed in Table 4. Figure 2 graphically represents the percentage distribution of these four types of extended common coupling.

Table 4. Number of instances of `current` that induce different types of extended common coupling in Linux kernel modules

Type	A	B	C	D	Total
Definition	114	0	0	0	114
Use	223	58	55	46	382
Total	337	58	55	46	496

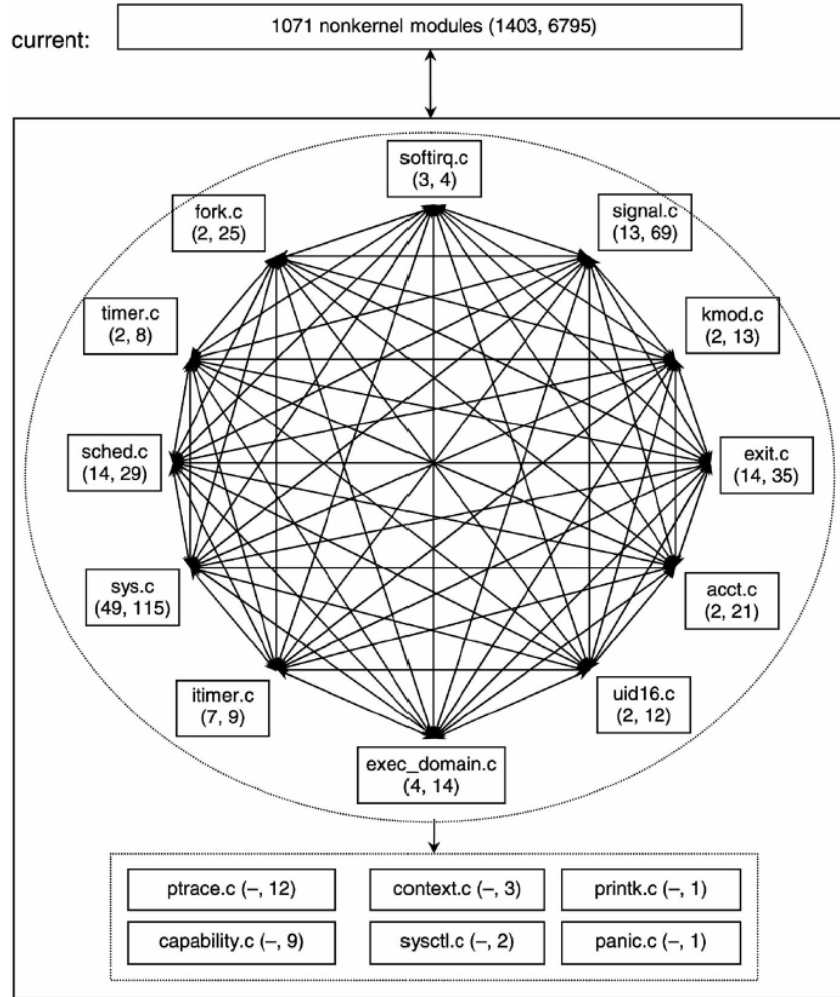


Figure 1. The definition and use of global variable **current** in version 2.4.20 [6].

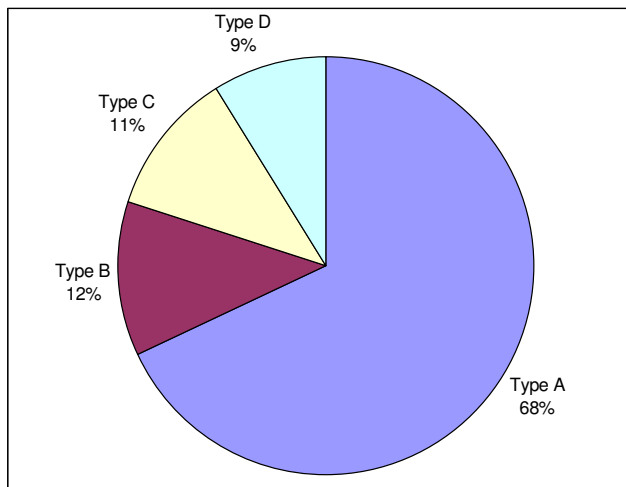


Figure 2. The percentage distribution of the four types of extended common coupling induced by **current** in Linux kernel.

The result shows that Type A (stamp-common coupling) is the major type (about 68 percent) of extended common coupling induced by global variable **current** in Linux kernel. About 12 percent instances of **current** induce extended common coupling Type B. Type C and Type D extended common coupling induced by **current** in kernel modules are about 11 percent and 9 percent respectively. As expected, there are no instances of “pure” common coupling. Both Type-C and -D extended common coupling are associated with control coupling. The value of **current** in Type C or D is used to control the internal logic of the program. There are 101 instances of **current** inducing Type C or Type D, which accounts about 20 percent of the total instances of common coupling. This means changes made to the value of **current** in one module may affect up to 101 instances of internal logic in other kernel modules.

Type A and Type C are both associated with stamp coupling and Type B and Type D are both associated with data coupling. A summary of these types is shown in Table 5.

Table 5. Summary of instances of current by the type of extended common coupling they induce

	Type A and Type C	Type B and Type D	Total
Number of instances	392	104	496
Percentage	79%	21%	100%

Table 5 shows that about 79 percent of instances of `current` are referenced by some fields of its structure. About 21 percent of instances of `current` are referenced to the whole data structure. Because stamp coupling is stronger than data coupling, the interaction of common coupling with stamp coupling is more complicated than the interaction of common coupling with data coupling. More instances of `current` belonging to either stamp-common coupling or stamp-control-common coupling means, in most cases, only some fields of `current` are referenced, instead of the whole data structure. This has two negative effects on maintenance of Linux:

- 1 More data information than necessary is passed and manipulated between modules. This makes the source code more difficult to understand. For example, module A only needs the information of `current->state` from module B, but the whole `current` is available to A. Therefore, it needs more effort to understand the source code and to make the correct changes.
- 2 More data information than necessary exposed to other modules is likely to result incorrect changes on `current` and may bring more chances of regression faults. Therefore, more effort is needed to prevent the faults, detect the faults, and correct the faults.

Our study is performed on a special global variable — `current` in Linux kernel modules. As discussed in [6], `current` represents the complication of common coupling in Linux. There are 1403 definitions of `current` in nonkernel modules (Figure 1). The interaction of these instances of `current` with stamp coupling will make the maintenance more difficult.

4. CONCLUSIONS

In this study, we classified each instance of a global variable according to the various forms of coupling it induces. We call this kind of classification “extended common coupling.” We studied each instance of global variable `current` in the Linux kernel and determined the type of extended common coupling it induces. Our study shows that, in most cases, `current` is used with stamp coupling instead of data coupling, which makes Linux more difficult to maintain.

5. ACKNOWLEDGMENTS

The authors would like to thank Prof. Schach and Prof. Fisher for their many suggestions.

6. REFERENCES

- [1] Stevens, W. P., Myers, G. J., and Constantine, L. L. Structured design. *IBM Systems Journal*, 13, 2, 1974, 38–54.
- [2] Jones, P. *The Practical Guide to Structured System Design*, Yourdon Press, New York, 1980.
- [3] Offutt, J., Harrold, M. J., and Kolte, P. A software metric system for module coupling. *Journal of Systems and Software*, 20 (March, 1993), 295–308.
- [4] Schach, S.R., *Object-Oriented and Classical Software Engineering*, Sixth Edition, McGraw-Hill, New York, 2005.
- [5] Schach, S. R., Jin, B., Wright, D. R., Heller, G. Z., and Offutt, J. Quality impacts of clandestine common coupling. *Software Quality Journal*, 11 (July 2003), 211–218.
- [6] Yu, L., Schach, S. R., Chen, K., and Offutt, J. Categorization of common coupling and its application to the maintainability of the Linux kernel. *IEEE Transactions on Software Engineering*, 30, 10, 2004, 694-706.
- [7] Rusling, D. The Linux kernel, 1999, www.linuxhq.com/guides/TLK/tlk.html.

APPENDIX A. THE GENERAL TYPE OF EXTENDED COMMON COUPLING

In our case study of global variable `current` in the Linux kernel, we found four types of extended common coupling: stamp-common coupling, data-common coupling, stamp-control-common coupling, and data-control-common coupling. Here, we discuss theoretically the general types of extended common coupling.

Common coupling must be associated with a global variable. This global variable could be a simple data type or a composite data structure. If the variable is a simple data type, the two modules that access the global variable are considered data-common coupled; if the variable is a composite data structure, the two modules are considered as either stamp-common coupled or data-common coupled, depending on whether the module operates on one or more fields or the whole data structure. Because a global variable is referenced by both modules, every instance of this global variable will induce either stamp coupling or data coupling between the two modules. So, the simplest form of extended common coupling is in one of two exclusive forms, “stamp-common coupling” and “data-common coupling.”

If a global variable is used to control the internal logic of a module, this module is also considered control-coupled with the other module that accesses this global variable. Combining the two simplest forms of extended common coupling with control coupling, we have the other two types of extended common coupling: stamp-control-coupling and data-control-common coupling.

Now, we have four types of extended common coupling: stamp-common coupling, data-common coupling, stamp-control common coupling, and data-control common coupling. These are all the possible types of extended common coupling induced by a global variable by considering data coupling, stamp coupling, control coupling, and common coupling.

Another coupling is content coupling. An example of content coupling is module A branches to a local label of module B. Because content coupling defies the principle of modularity, this is not supported by C/C++. However, this can be done in other languages, like Ada and assembler. For languages that support branches between modules, it is possible that content coupling maybe associated with common coupling. In the following example (written in pseudocode), suppose Function1 is in module A and Function2 is in module B, and that both A and B access global variable gv.

```
int Function1 (int a)
{
    printf ("In Function1\n");
    a += 2;
    if (gv > 0) goto F2A;

    return a;
}

void Function2 (void)
{
    printf("In Function2\n");
```

F2A:

```
    printf("At Function2A\n");
}
```

In module A, the instance of gv induces common coupling, data coupling, and control coupling between module A and module B. By evaluating the value of gv, module A may branches to module B. gv also induces content coupling between A and B. We may call this kind of extended common coupling “data-control-content-common coupling.” Alternatively, we may have “stamp-control-content-common coupling.” So, in theory we could have more types of extended common coupling. Depending on the implementation language, we may have other types of extended common coupling, such as “stamp-content-common coupling,” and “data-content-common coupling.” In summary, theoretically analyzed, more types of extended common coupling induced by a global variable may exist between modules by considering data coupling, stamp coupling, control coupling, and content coupling. However, because content coupling defies the principle of modularity, in practice, only four types of extended common coupling are widely used. They are stamp-common coupling, data-common coupling, stamp-control common coupling, and data-control common coupling. This was verified by our case study of the instances of global variable current in the Linux kernel.