Operating System Process Management and the Effect on Maintenance: A Comparison of Linux, FreeBSD, and Darwin

Liguo Yu
Computer Science and Informatics
Indiana University South Bend
South Bend, IN 46634, USA
ligyu@iusb.edu

ABSTRACT. Process management is one of the most important and relevant tasks in operating system design. In this paper, we investigate the process management in Linux, FreeBSD, and Darwin. We compare the data structures used to represent process and the global variables used to control the current active process in three operating systems. Based on the definition-use analysis, we study how the number of instances of process control global variable can affect the maintenance of the operating system kernel. This effect is demonstrated in an empirical study in the relationship between the number of kernel lines of code modified and the number of instances and number of definitions of process controller global variable. We conclude that the way process management implemented in Linux makes it more difficult to maintain than FreeBSD and Darwin.

Keywords: Modularity, common coupling, maintenance.

(Received January 19, 2006 / Accepted May 10, 2006)

1. Introduction

Coupling is a measure of the degree of dependency between two software components (classes, modules, packages, or the like). A good software system should have high cohesion within each component and low coupling between components. There are several different coupling categorizations [1], [2], [3], all of which include *common coupling* (two modules are common coupled if they access the same global variable). Common coupling is considered to be a strong form of coupling, that is, it induces a high degree of dependency between software components, making the components difficult to understand and maintain [4].

Coupling between components strengthens the dependency of one component on others and increases the probability that changes in one component may affect other components, which makes maintenance difficult and likely to introduce regression faults [5], [6]. Coupling has not yet been explicitly shown to be related to maintainability. However, it has been shown that coupling is related to fault-proneness of a software system [6], [7], [8]. If a module is fault-prone then it will have to undergo repeated maintenance, and these frequent changes are likely to compromise its maintainability. Furthermore, these frequent changes will not always be restricted to the fault-prone component itself; it is not uncommon to have to modify more than one component to fix a single fault. Consequently, the fault-proneness of one component can adversely affect the maintainability of a number of other components. In other words, it is easy to believe that strong coupling can have a deleterious effect on maintainability.

In previous research, we studied common coupling in kernel-based software (such as operating systems) and categorized global variable in terms of the possible impact a change to it would have on the kernel [9]. The most deleterious form of common coupling is category-5.

In operating systems, in order to achieve high efficiency, the process management is usually implemented via a global variable that accesses all the current active processes in the system. As further discussed in Section 3, this global variable is a category-5 variable. In this paper, we investigate the role played by this category-5 global variable with regard to the maintainability of three open-source operating systems: Linux, FreeBSD, and Darwin.

The remaining of the paper is organized as follows: Section 2 outlines the categorization of common coupling in kernel-based software. We discuss operating system process management in Section 3. In Section 4, we describe Linux, FreeBSD, and Darwin. We compare the process management of Linux, FreeBSD, and Darwin in Section 5. Section 6 contains the empirical study of the correlation between the maintenance effort and the process control global variable. Section 7 contains the discussions, conclusions and future research.

2. Categorization of common coupling

Each occurrence of a variable in source code is either a definition of that variable or 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 = 10. The *use* of a variable x is a statement that utilizes the value of x, such as y = x – 9. From the creation of a variable to the destruction of that variable, each time the variable is invoked, it is either assigned a new value (a definition) or its present value is used (a use).

Many software products, especially operating systems and database management systems, comprise a kernel, a set of components common to all installations, together with a set of architecture-specific or hardware-specific nonkernel components. We refer to a software product that is comprised of common kernel components together with optional nonkernel components as *kernel-based software*.

The kernel is the most important part of a kernel-based software product. Therefore, the maintainability of the kernel reflects the maintainability of the kernel-based software product. Common coupling within a kernel-based product increases the dependency of the kernel on other components and, therefore, decreases the maintainability of the kernel.

From the viewpoint of maintenance, changes to a definition of a global variable can affect the use of that global variable, but not vice versa. In previous study [9], we used definition-use analysis to study global variable and presented an ordered categorization of common coupling within kernel-based software. Global variables are divided into five categories, from the least deleterious (category-1) to the most harmful (category-5). For example, a category-1 global variable is defined in kernel components but has no uses in kernel components. Because there is no use of a category-1 global variable in a kernel component, definitions in other components (kernel or nonkernel) cannot affect kernel components. Consequently, all kernel components are independent with respect to this global variable, and the presence of a category-1 global variable will not cause difficulties for kernel component maintenance.

On the other hand, a category-5 global variable is defined in both kernel components and nonkernel components, and is used in kernel components. A kernel component that uses a category-5 global variable is therefore vulnerable to a modification made in a kernel component or a nonkernel component in which that global variable is defined. It is extremely difficult to minimize the impact of changes that involve category-5 global variables. Therefore, category-5 global variable has potential effect on the maintenance of kernel modules. (For details of global variable categories 2, 3, and 4, the reader is referred to [9].)

3. Process management in operating systems

In operating systems, process is defined as "A program in execution" [10]. Process can be considered as an entity that consists of a number of elements, including: identifier, state, priority, program counter, memory pointer, context data, and I/O request. The above information about a process is usually stored in a data structure, typically called *process block*. Figure 1 shows a simplified process block [10]. Because process management involves scheduling (CPU scheduling, I/O scheduling, and so on), state switching, and resource management, process block is one of the most commonly accessed data type in operating system. Its design directly affects the efficiency of the operating system. As a result, in most operating systems, there is a data object that contains information about all the current active processes. It is called *process controller*. Figure 2 shows the structure of a process controller [10], which is implemented as a linked-list of process blocks.

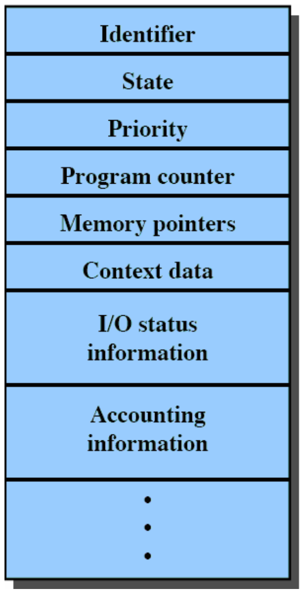

Figure1: Simplified process block [10]

In order to achieve high efficiency, process controller is usually implemented as a global variable that can be accessed by both the kernel modules and nonkernel modules. For example, any time a new process (task) is created, the module that created this process should be able to access the process controller to add this new process. Therefore, process controller – the data object that controls the current active process – is usually implemented as a category-5 global variable. This means, both the kernel modules and nonkernel modules can access process controller to change its fields and these changes can affect the uses of process controller in kernel modules.

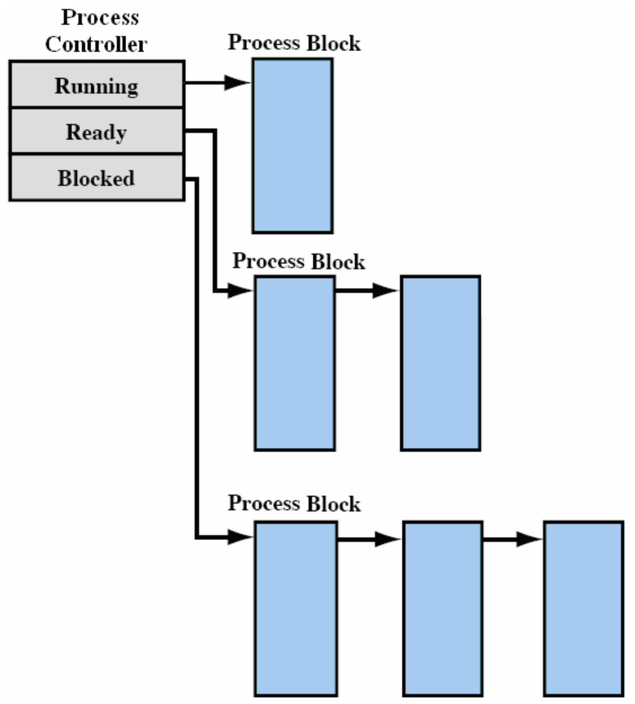

Figure2: Process controller structure [10]

4. Linux, FreeBSD, and Darwin operating systems

Linux, FreeBSD, and Darwin are three open-source operating systems. Linux is a completely new implementation of UNIX using module structure. The advent of KDE (K Desktop Environment) and GNOME (GNU Object Model Environment) makes Linux a user-friendly desktop operating system. FreeBSD is another widely used BSD (Berkeley Software Distribution) operating system. It is also a UNIX-like operating system. It is well suited for both desktop and server applications. It features high performance file system operations, and provides robust network services. Darwin is an open-source core used in Apple OS X. It consists of two major components: a microkernel based on Mach, and a full implementation of BSD (largely based on FreeBSD).

All three open-source operating systems are kernel-based, which means they contain both architecture independent kernel modules and architecture dependent nonkernel modules. In this paper, we studied Linux 2.4.20, FreeBSD 5.1, and Darwin XNU-517. The size of three operating systems is shown in Table 1. All three operating systems are written in C. Each ".c" or ".h" source file is considered as a module.

Table 1: The kernel and nonkernel structure of three operating systems

Operating system	Kernel modules	Nonkernel modules	Kernel KLOC	Total KLOC
Linux 2.4.20	26	9,407	14	4,260
FreeBSD 5.1	131	3,353	108	1,793
Darwin XNU-517	196	1,656	110	744

5. Process management in Linux, FreeBSD, and Darwin

Table 2 summarizes the process management structure of Linux, FreeBSD, and Darwin. In Linux, process block is implemented as a data structure, task_struct. In FreeBSD, process block is implemented as a data structure, proc. In Darwin, process block is implemented as a data structure, task. All three data structures contain the similar information about a process, such as process id, process state, file information, and so on. The process block (task_struct in Linux, proc in FreeBSD, and task in Darwin) is the most complicated data structure in their corresponding operating systems. For example, task_struct contains 83 field variables; 60 are primitive types, 3 are composite data structures, and 20 are pointers to composite data structures [11].

In Linux, process controller is implemented as a global variable, current. In version 1.0.9, current was declared as a pointer to data structure task_struct in kernel module sched.c:

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

From version 1.3.31 onward, current was declared as a preprocessor macro get_current(), which is an inline function that returns a pointer to data structure task_struct. In both cases, current can be viewed as a pointer to data structure task_struct.

In FreeBSD, global variable curproc is used to represent the process controller. In module proc.h, it is declared as a pointer to data structure proc:

```
struct proc* curproc;
```

In Darwin, global variable kernel_task is used to represent the process controller. In module task.c, it is declared as a pointer to data structure task:

```
typedef struct task  *task_t;
task_t  kernel_task;
```

Using LXR (Linux Cross Reference) tool, for each operating system, we determined all the occurrences of the process controller global variable (current for Linux, curproc for FreeBSD, and kernel_task for Darwin) in kernel modules and in nonkernel modules. For each instance of the global variable, we determined whether it is a definition or a use. The definition-use analysis was performed on the basis of the theory outlined in Section 2. For example, the statement

```
current->state = TASK_RUNNING;
```

was considered a definition of current, because the value of current (or, more precisely, the data structure to which it points) is changed. Conversely, the statement

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

was considered a use of curproc, because the value of curproc (or, more precisely, the data structure to which it points) is referenced, but not changed. On the other hand, the statement

```
kernel_task->request_count ++;
```

was considered both a definition and a use of kernel_task, because the value of kernel_task (or, more precisely, the data structure to which it points) is first referenced and then changed.

Table 3 summarizes the general results of definition-use analysis. In Linux kernel modules, there are 114 instances of definitions and 382 instances of uses of current. In nonkernel modules, current is defined 1,403 times and used 6,795 times. Adding the definitions and the uses yields a total of 8,694 instances of current in Linux. The corresponding total number of instances of curproc in FreeBSD and kernel_task in Darwin are 483 and 104 respectively. The number of instances of process controller global variable in Linux is about 18 times of FreeBSD and 84 times of Darwin. Figure 3 shows the number of instance of global variable per KLOC (thousand of lines of code). We can see, considering the size difference of three operating systems, Linux still has more instances of process controller global variable than FreeBSD and Darwin.

Table 2: Process management in three operating systems

Operating system	Process block		Process controller	
	Name	Description	Name	Description
Linux	task_struct	Composite data structure	current	Pointer to task_struct
FreeBSD	proc	Composite data structure	curproc	Pointer to proc
Darwin	task	Composite data structure	kernel_task	Pointer to task

Table 3: Definitions and uses of the process controller global variable in three operating systems

Operating system	Global variable	Kernel modules		Nonkernel modules		Overall	
		Number of definitions	Number of uses	Number of definitions	Number of uses	Number of definitions	Number of instances
Linux	current	114	382	1,403	6,795	1,517	8,694
FreeBSD	curproc	22	95	3	363	25	483
Darwin	kernel_task	5	46	1	52	6	104

Each installation of Linux, FreeBSD, or Darwin consists of all the kernel modules, plus a set of nonkernel modules specific to that installation, its architecture, and its drivers. It might therefore be argued that, in any one installation, the number of instances of process controller global variable (current for Linux, curproc for FreeBSD, and kernel_task for Darwin) in nonkernel modules is likely to be far smaller. From the viewpoint of maintenance, however, what is important is the total number of instances of the process controller global variable. If a change is made to a global variable, it has to be consistently made to every instance of that global variable. Thus, the total number of instances is what counts, not the number in a specific installation.

As described in Section 2, the instances of global variable that can affect kernel are the definitions. Because any changes to a definition can result the corresponding changes to the use of the global variable in kernel. From Table 3, we see that there are 114 instances of definitions of current in kernel modules, and 1,403 instances of definitions in nonkernel modules. That is, there are 1,517 instances of definitions of current that could affect a kernel module if a modification were made to the module containing that definition of current. The corresponding number of definitions of curproc in FreeBSD and kernel_task in Darwin are 25 and 6, which are much smaller than the number of current.

Figure 4 compares the number of definitions of process controller global variable per KLOC of the three operating systems. Because every definition of the process controller global variable constitutes a potential source of vulnerability from the viewpoint of maintenance of the operating system kernel, Figure 4 shows that changes to current in Linux are likely to need more effort than changes to curproc in FreeBSD and changes to kernel_task in Darwin.

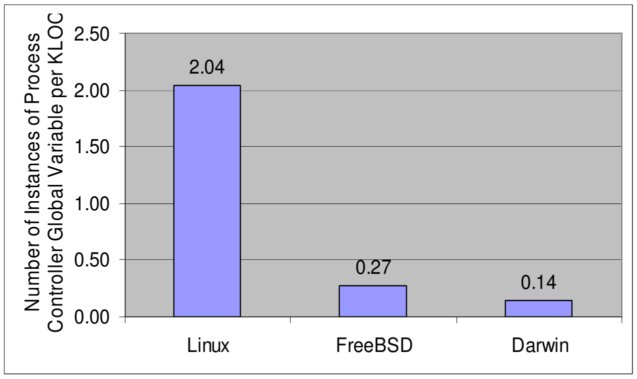

Figure 3: Comparisons of Linux, FreeBSD, and Darwin: number of instances of process controller global variable

6. Correlation between maintenance effort and process controller global variable

In Section 2, we analyzed the relationship between category-5 global variable and the maintenance of kernel modules. Process controller is a category-5 global variable. Our analysis indicates that more instances of process controller global variable can affect the maintainability of kernel modules, which in turn can result more maintenance effort. To understand their relationship empirically, we performed the following study.

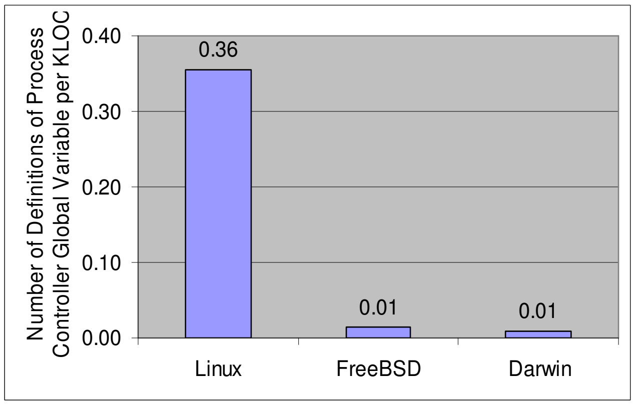

Figure 4: Comparisons of Linux, FreeBSD, and Darwin: number of definitions of process controller global variable

From version 1.0.0 to version 2.4.20, we studied 299 release of Linux, for each release, we determined the number of instances and the number of definitions of process controller global variable current. For each release, we also determined the number of kernel lines of code modified compared to the previous version, which is used to represent the maintenance effort. We would expect to find the maintenance effort (the number of kernel lines of code modified) increases as the number of instances of current increases and the number of definitions of current increases. In more detail, we tested the following two null hypotheses:

- H_{01}: There is no linear relationship between the number of kernel lines of code modified and the number of instances of current in each release.
- H_{02}: There is no linear relationship between the number of kernel lines of code modified and the number of definitions of current in each release.

In these tests, the number of kernel lines of code modified is the dependent variable Y, the number of instances of current and the number of definitions of current are identified as independent variables X.

To test these hypotheses, we would need to calculate the correlation, which summarizes the strength of the relationship between the two variables X and Y. Several different correlation coefficients have been put forward, including Pearson's correlation coefficient and Spearman's rank correlation coefficient [12]. For Pearson's correlation coefficient to be valid, variables X and Y both need to be normally distributed. However, it is unlikely that either X or Y will have a normal distribution. Therefore, we use Spearman's rank correlation test. If the rank correlation coefficient proves to be statistically significant at, say, the 0.01 level, we will reject the null hypothesis, and accept the alternate hypothesis.

Table 4 shows the correlation coefficients between maintenance effort (the number of lines of code modified) and the two measures of current, and the corresponding p-values. Both two tests show the correlation coefficients are significant at the 0.01 level (2-tailed). Therefore, we reject the two null hypotheses. Because we use the number of kernel lines of code modified to represent the maintenance effort of kernel modules, we conclude that

- There is significant positive linear correlation between the maintenance effort of kernel modules and the number of instances of process controller global variable.
- There is significant positive linear correlation between the maintenance effort of kernel modules and the number of definitions of process controller global variable.

It should be noted that the hypothesis tests here did not show the causal relationship between the number of instances of process controller global variable and the maintenance effort. It only provided the empirical evidences. To show the causal relationship empirically, a well organized experiment should be performed, in which, all other factors must be fixed.

Table 4: The correlation between the number of kernel lines of code modified and the two measures of current

Measure	Number of instances of current	Number of definitions of current
Correlation coefficient	0.151	0.307
Number of data set	299	299
P-value	<0.009	<0.001

7. Discussions, conclusions, and future research

Our empirical study on 299 versions of Linux shows that strong linear correlations exist between kernel maintenance effort (the number of kernel lines of code modified) and the number of instances and the number of definitions of process controller global variable, which statistically indicate the relation between maintenance effort and process control global variable.

Process controller is usually designed as a category-5 global variable. This has been verified in Linux, FreeBSD, and Darwin. However, the ways to implement the global variables are different for three operating systems, which have different degree of effects on kernel maintenance. Our study shows current has more deleterious effects on Linux than curproc on FreeBSD and kernel_task on Darwin.

Linux is continuously growing with more drivers being added and more platforms are supported. Adding more drivers means more processes will be associated with global variable current. If more platforms are supported, more platform-specific tasks will be added, too, causing further instances of current to be added. This will result in even greater increases in the number of instances of current. That is, as Linux grows, the kernel maintenance problem caused by current will be exacerbated.

To summarize, in this paper, we compared the process management in Linux, FreeBSD, and Darwin. In all three operating systems, process control is managed via a global variable, a pointer to a composite data structure that stores the process information. However, we found, the number of instances of process controller global variable current in Linux is very different from curproc in FreeBSD and kernel_task in Darwin. Based on the definition-use analysis of its effect on maintenance and an empirical study on the relationship between maintenance effort and process controller global variable, we deduce that Linux will be more difficult to maintain than FreeBSD and Darwin.

Our future research will study the architecture difference among Linux, FreeBSD, and Darwin to understand how the system architecture contributes to the difference in the number of instances of process controller global variable. Based on this, we will study how Linux should be restructured to improve its maintainability.

References

[1] Stevens, W. P., Myers, G. J., and Constantine, L. L. *Structured design*, *IBM Systems Journal*. v. 13, no. 2, p.115–139, 1974.

[2] Offutt, J., Harrold, M. J., and Kolte, P. *A Software metric system for module coupling*, *Journal of Systems and Software*. v. 20, p. 295–308, 1993.

[3] Jones, P. *The Practical Guide to Structured Systems Design.* Yourdon Press, New York, 1980.

[4] Schach, S. R., Jin, B., Wright, D. R., Heller, G. Z., and Offutt, J. *Quality impacts of clandestine common coupling*, *Software Quality Journal*. v. 11, p. 211–218, 2003.

[5] Briand, L. C., Daly, J., Porter, V., and Wüst, J. *A comprehensive empirical validation of design measures for object-oriented systems*, *Proceedings of the 5th International Software Metrics Symposium*, Bethesda, MD, p. 246–257, 1998.

[6] Troy, D. A. and Zweben, S. H. *Measuring the quality of structured designs*, *Journal of Systems and Software*. v. 2, p. 112–120, 1981.

[7] Kafura, D. and Henry, S. *Software quality metrics based on interconnectivity*, *Journal of Systems and Software.* v. 2, p. 121–131, 1981.

[8] Selby, R. W. and Basili, V. R. *Analyzing error-prone system structure*, *IEEE Transactions on Software Engineering*. v. 17, p. 141–152, 1991

[9] 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.* v. 30, p. 694–706, 2004.

[10] Starllings, W. *Operating Systems: Internals and Design Principles*, 5ed, Prentice Hall, 2004.

[11] Rusling, D. The Linux kernel. 1999, www.linuxhq.com/guides/TLK/tlk.html

[12] Nolan, B. *Data Analysis, an Introduction*, Polity Press, Cambridge MA, 1994.