A Study on Operating System Optimization

Tooru Terada

School of Information Science,
Japan Advanced Institute of Science and Technology

February 14, 1997

Keywords: Extensible operating system, Dynamic code generation, Module.

The current requirements of applications for operating systems are very complex. Thus, it is impossible for any single operating systems to satisfy all requirements of applications. So, today's researches are directed to the systems which can be extensible. The microkernel architecture is one of the extensible system models. It moves several services from traditional monolithic kernels to user-level servers. It increases the flexibility of operating systems because of its portability of servers. In addition, it increases the safety that applications are never influenced by servers which they does not use.

However, operating systems based on the microkernel architecture have a drawback concerned with their performance. They need much protection domain crossing than traditional monolithic kernels because of the communications with servers. Several approaches to improve this drawback are proposed. The approach moving application's functionalities into the kernel seems to be effective. In systems using the approach, applications load the code into the operating system kernel dynamically in order to alter its behavior. This reduces the number of context switches. In addition, downloading the code can be used to create data paths in the kernel. This removes the need for much of data copies across the user/kernel boundary. If applications load the code into the kernel, safety issues should be taken into account. Loading codes should not expose other applications in danger. However, we require much flexibility or the better performance so that it is difficult to design codes with no errors. Application designers which include downloading code must take care not to include some errors, since there is no mechanism to check completely not only syntactic errors but also semantic errors of kernel extensions. To design downloading code with no error, modular designs are expected. In addition, modular softwares are reusable. However, modular softwares have worse performance than monolithic and specific software. This thesis proposes optimizations improving this drawback of modular programming.

One of the cause of the poor performance in modular softwares is that many modules designed individually manipulates the same data segment respectively. Usually, respective

modules load data to registers, manipulate them, and store them to memories. Loading and storing data in each modules are redundant. To reduce this redundancy, this thesis provides a mechanism and a set of application programming interfaces which integrates respective data manipulations into pairs of load/store. This optimization is effective if the same data is manipulated repeatedly by many modules.

Another cause of poor performance in modular softwares arises when a set of modules is triggered the execution by some outer events such as in-kernel events. In the middle of executing, there are some cases that modules conclude from their execution states that no benefit is brought to the application by that execution. Then, they abort the execution. If many modules are executed until they abort, it includes many useless execution. In another case, even if the execution is benefitable for the application, executing same judge at many module is redundant. Now, this thesis introduce the filter into module programming. This filter is similar to the packet filter in network. Each modules pass their filters to the modules being executed formerly. Receiving modules collects and executes their conditional evaluation. This reduces useless or repeated execution. Moreover, if receiving modules need not execute their conditional evaluation, they are let propagate filters to more former modules. This technique brings not only cuts unnecessary executions but also improvement in memory locality. Improvement in memory locality can reduce cach misses.

These optimizations can be implemented with dynamic code generation. Dynamic code generation is the creation of executable code at runtime. It is a powerful technique, enabling applications to use runtime information to improve performance by up to an order of magnitude. Dynamic code generation is useful for our optimization. Firstly, dynamic code generation reduce procedure call by composing set of modules into a function. Secondly, since it can decides instruction order dynamically, it integrates data manipulations in order to reduce load/store between memorys and registers. Finally, it compiles filter structures to machine code in order to reduce execution costs of composed function.

We implemented a prototype system including these optimization on Real-Time Mach kernel. Real-Time Mach is a real-time extension of the Mach operating system. We also choose VCODE system which has developed at MIT as a dynamic code generating system. The VCODE system is a set of C macros and support functions that allow programmers to portably and efficiently generate code at runtime. VCODE generates code in place. It eliminates the need to build and consume an intermediate representation. VCODE instructions are translated directly to the machine instructions that they correspond to. This allows us to directory construct arbitrary code at runtime, such as integrated data manipulation and compiled filter structure.

We show the effectiveness of our proposal by showing some basic experiments. In addition, we show the effectiveness of downloading code to kernels or servers.