A Customized Linux Kernel for Providing Notification of Pending Financial Transaction Information

J. Montgomery¹, Gregory B. Brewster², Wai Gen Yee³
¹Information Retrieval Laboratory, Illinois Institute of Technology, Chicago, IL 60616 USA
²School of Computing, DePaul University, Chicago, IL 60604 USA
³Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616 USA
¹jmontgomery@ir.iit.edu, ²gbrewster@cdm.depaul.edu, ³yee@iit.edu

Abstract—Financial transaction systems, such as those utilizing the Financial Information Exchange (FIX) protocol [1], must make buy/sell decisions based on information received over a network interface. However, at the time that a decision to buy or sell is made, some data on the system may not yet be visible to the financial engine software because it has not yet completed being processed by the network protocol stack. This data is hidden from a FIX engine running in user space until the network protocol processing (in the kernel) is fully completed and the message contents are copied from kernel space to user space. This hidden data may contain information that would lead to a more advantageous transaction if it were available to the financial application at the time that the trade decision is made.

We have made several modifications to the TCP/IP protocol stack in the Linux operating system. Our protocol modifications allow a user application to determine if there are pertinent pending messages currently being processed in the kernel. If this is the case, the user application may then choose to delay the buy/sell decision by a short time until these additional messages have completed kernel processing and become available to the user application.

We have implemented this mechanism under Linux 2.6.27.24 and are evaluating alternate implementations and the significance of the decision improvements that result from this enhancement.

Keywords-component; TCP/IP; Linux, network protocol stack

I. INTRODUCTION

Modern financial transaction systems depend on messages exchanged across high-speed networks [2][3]. At any given point in time, these financial systems, running in user application space, must make buy/sell decisions based on the buy/sell offers received so far from the network. However, data packets arriving on the network interface are not immediately available to the financial transaction software, as they must first undergo protocol processing by the operating system software running in “kernel space”.

In current systems, no information about kernel-space messages is available to applications executing in user space until all protocol processing is complete and these messages have been copied from kernel memory to user memory.

Figure 1 illustrates the flow of messages from the network interface through the IP and TCP/UDP network layers in kernel space, and then into the user space storage associated with the financial engine application via the read() function call using the sockets interface. Message data is not visible to the financial engine until it has been copied over to user space.

This “hidden” transaction data may contain information that could be used to improve a current trade for a particular security. For example, one of these hidden messages may contain a higher buy offer or a lower sell offer than those currently in the book.

Using FIX protocol messages as the focus for our current research, we implemented Linux kernel-level code that provides the following services:

1) a Pending Messages indicator which provides a user application a means to determine whether there are FIX messages currently being processed in kernel space that have not yet been made available for reading by user financial applications.

2) a read_pending service that provides a means for a user application to be delayed briefly until previously identified pending messages have been read and processed before making a trade decision.

Although such modifications may slightly delay (~20 µsecs) the execution time of a pending trade, this delay will occur only if the application has verified that there is additional information resident on the trading system that will shortly be available to the application. We emphasize that the additional information gained by this delay was already on the trading
system at the original trade decision time, but was simply not yet available to the financial application at that time.

To ensure that our Linux kernel modifications are easily maintained across system patches and updates, we implement them as Linux Kernel Loadable Modules (KLMs) [4], which may be installed over any compatible version of the base Linux operating system code.

We believe that the use of these services can offer enhanced financial performance, as they improve the quantity of information available at the decision time.

II. IMPLEMENTATION

A. Operating System Changes

There are several methods by which the operating system may notify the financial application that there are FIX messages currently being processed in kernel space. The simplest may be to simply make a FIX flag available to the application, which stores a simple Boolean value of true if FIX messages are currently being processed and false if not. This flag variable would be set by the operating system and would be available to the user application on a read-only basis.

Implementation of such a flag variable is straightforward, but it presents a difficulty to the user FIX engine – it cannot tell how many messages are being processed, so it does not know how many FIX messages can be read before all information is available for the current trade decision.

An enhancement to the flag implementation is the FIX kernel message count implementation, where the operating system makes an integer count of the number of FIX messages currently awaiting processing through the network stack in kernel. Again, this is implemented as an integer value that is set by the operating system and is available to the user application via protected fixed memory. Figure 2 shows this new count variable.

The operating system updates required to maintain this new FIX Kernel Message Count variable are straightforward. The variable is initialized to zero at system initialization. The IP code that first processes messages coming off the network interface is modified to check each incoming message to determine if it is a FIX message (which is determined via the port numbers in the TCP/UDP header) and increment the message count variable if so. The socket read() code that copies a message from kernel to user space is modified to decrement the message count variable for each FIX message that it copies into user space.

This simple implementation assumes that there is only one FIX user application executing and that all FIX messages are of interest. It is also possible to modify this implementation such that the counter variable will only be incremented for a subset of FIX messages specified by user-selectable criteria, such as source IP, source port, or destination port. If the FIX messages have not been compressed or encoded by technologies such as FAST, the user application may also be able to specify particular FIX parameters, such as specific commodities that are of interest. The operating system may be modified to count only messages that match these criteria, as illustrated in figure 3.

B. User Application Changes

To take advantage of the operating system modifications described above, FIX applications can be modified to read the FIX Kernel Message Count variable and utilize it to make decisions on when to execute financial transaction decisions. A straightforward implementation would be for the FIX application to read the Kernel Message Count variable when it is ready to make a buy/sell decision and then wait until this number of additional FIX messages have been read and processed before finalizing the decision.

III. RESULTS AND FUTURE WORK

We have implemented the FIX Kernel Message Count functionality as a Kernel Loadable Module on Linux 2.6.27.24 running under Parallels™ 4.0 (a virtual machine for operation systems). Our implementation validates the feasibility of these operating system modifications.

The next step is to feed historical options/trade data to our system and analyze the value-benefit of delaying a trade until all buy/sell messages can be used in the decision process.

REFERENCES