A CTRON Kernel Benchmark Program

Kazufumi Yamamoto Shin Futagami Toshikazu Ohkubo Hisayoshi Kurosawa

NTT Communication Switching Laboratories Mitsubishi Electric Corporation

Abstract

The software used in systems for information, communication, and switching processing can be more portable and easier to maintain when these systems used layered software structures (platforms). It is therefore necessary to establish techniques for comparatively evaluating the performance of different platforms designed in conformance to the interface specifications for each layer of these hierarchical systems. This paper proposes a method for using a benchmark program, to quantitatively compare kernel interface performances, which are central to the performance of CTRON-specification systems.

1. Introduction

A platform is the common base on which system-specific application software is executed. The provision of platforms, by organizing various kinds of systems into layers, is becoming common as a means of enhancing software portability and simplifying software maintenance tasks. Platforming allows a system developer to select from among the platforms provided by multiple vendors according to the performance, cost, and reliability required for each particular system. The platform approach is important not only in information processing fields such as those based on UNIX, but also in switching and communication fields with their large-scale software. The present paper adopts the CTRON model for hierarchical realtime systems in the communication processing field and proposes a method for using a benchmark program to quantitatively compare the performance of different platforms conforming to CTRON interface specifications.

2. Evaluation Object

For consideration in terms of the portability and extendibility of software the CTRON model can be roughly classified into the software in the platform constituting the basic OS interface and into the upper software used in common by two or more platforms (see Figure 1). The total performance of a CTRON specification system is determined by the way it uses the upper software and the way its platform performs. Use of the upper software uses a platform through the basic OS interface, and the platform performance consists of hardware performance, compiler performance, and the basic OS implementation performance. The general performance, however (including that of the upper software) can be compared by measuring the total execution time on the basic OS interface. We therefore created a standard benchmark program equivalent to common software and we could use to evaluate the total performance of a platform by measuring the execution time of the system call that constitutes the basic OS interface.

Figure 1. CTRON hierarchical model.
3. Platform Requirements

Realtime systems in switching and communication fields have the following characteristic features, which must be taken into account when setting the requirements of a platform that provides the running environment for application programs.

(1) Highly multiple realtime processing

Switching and communication processing systems are typically required to handle thousands or tens of thousands of tasks at the same time. Performance is therefore significantly affected by the task environment and method of implementing such functions as task communication. At the same time, realtime performance is required (whereby processing is completed within a required time frame) in order to satisfy such service quality factors as connection delay. To this end, many systems adopt scheduling mechanisms that make use of clock interrupt and preemption functions.

The platform must accordingly guarantee realtime performance, completing processing requests within a set time regardless of the number of tasks or other demands of the multiple processing environment.

(2) Input/output processing

Systems for switching and communication processing connect with a wide diversity of I/O equipment (communication controllers), and the access this equipment frequently. A common technique for reducing the overhead for monitoring all these I/O devices is to use a cyclic look-in method to detect I/O events.

Typically, the cyclic processing overhead varies with the number of connected devices, so the platform should be able to guarantee high throughput by efficiently executing processing requests from application programs while keeping down the fixed overhead from cyclic processing. (1)

4. Using Benchmark Programs to Evaluate Performance

4.1 Performance Indices

In evaluating the performance of a platform it is important to estimate system performance when application programs are run on the platform. Table 1 lists various factors used to define each aspect of system performance when applications are running. Especially when there are differences in the processor instruction set (CISC vs. RISC, for example), the traditional method of measuring performance by counting the number of dynamic steps does not apply (see Table 2). For this reason, when two or more platforms that differ in architecture are evaluated side by side, realtime response is an essential base for the performance evaluation method.

4.2 Features of Benchmark Evaluation

A benchmark program offers a simple way to compare the performance of systems with different architectures. Table 3 lists the features of three typical existing benchmark programs using integer calculations. If these programs are to be used effectively for quantitatively estimating platform performance when applications are running, we need

<table>
<thead>
<tr>
<th>Hardware performance</th>
<th>Instruction execution time, memory access time, response time of each I/O device, etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware characteristics</td>
<td>Instruction set (CISC/RISC, etc.), cache/TLB size and configuration, etc.</td>
</tr>
<tr>
<td>Software characteristics</td>
<td>Memory working set, instruction use distribution, dynamic steps, etc.</td>
</tr>
<tr>
<td>Compiler performance</td>
<td>Instruction compile ratio, memory access frequency, optimization effect, etc.</td>
</tr>
</tbody>
</table>
evaluation methods that make good use of these programs. Evaluation methods making use of these benchmark programs have the following advantages and disadvantages:

(1) Advantages
- Realtime-based performance can be evaluated.
- The program can be readily ported to different platforms.

(2) Disadvantages
It is often not possible to simulate application program operations fully. In particular, since the cache and TLB hit ratios depend on the application software, it is difficult to get an accurate measure of system performance by using only a benchmark program.

4.3 Benchmark Measurement Data
The way a platform performs while applications are running can be estimated by evaluating performance-defining factors like the three indicated below. The amount of processor resources usable by application programs is derived by measuring the total processor resources and by then subtracting both the fixed overhead for cyclic processing as well as the overhead needed to ensure that the CPU usage ratio has a performance margin that guarantees realtime characteristics. The number of application program running steps is the total number of steps in the processing part for the functional elements (like system calls) provided by the platform and in the processing that is application specific. The performance of the part for processing functional elements depends on the platform-specific performance, whereas the performance of the application-specific processing depends on the compiler and other aspects of the development environment, and on the characteristics of the application software. The degree to which each of these factors affects

Table 2. Example of execution time and the number of dynamic steps needed for different processors to execute system calls.

<table>
<thead>
<tr>
<th>System call</th>
<th>CISC processor</th>
<th>RISC processor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Execution time</td>
<td>DS</td>
</tr>
<tr>
<td>A</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>B</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>C</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 3. Features of various benchmark programs

<table>
<thead>
<tr>
<th>Item</th>
<th>EDN benchmark</th>
<th>Dhrystone</th>
<th>SPECint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Languages</td>
<td>Assembly, C, etc.</td>
<td>C, Pascal</td>
<td>C</td>
</tr>
<tr>
<td>Object of evaluation</td>
<td>Hardware</td>
<td>Hardware, Basic OS equivalent (UNIX)</td>
<td>Basic OS equivalent (UNIX)</td>
</tr>
<tr>
<td>Evaluates multitask environment</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Measures realtime characteristics</td>
<td>interrupt</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Evaluates I/O operations</td>
<td>No</td>
<td>No</td>
<td>file compress</td>
</tr>
<tr>
<td>Working set</td>
<td>300 bytes or less</td>
<td>1 KB or less</td>
<td>8 KB or less</td>
</tr>
</tbody>
</table>
performance varies with each platform, so it is not possible to judge the relative superiority of a platform only by evaluating certain individual parameters (see Figure 2). The CTRON benchmark program proposed here measures data from each of the following standpoints: and comprehensively evaluates system performance.

(1) Platform fixed overhead

This is the overhead for the various cyclic processing performed in the platform. Specifically, this overhead is determined by measuring the rate at which the processor is used by the cyclic scheduling mechanism based on clock interrupts, and by measuring processor use for the cyclic look-in processing of I/O devices.

(2) Platform-dependent part

The amount of processing for the various functional elements (like system calls) provided by the platform depends on the number of times each of the functional elements is used and on their execution time. Here the execution time for each functional element is measured, and the total execution time is calculated according to the number of times each element is used.

(3) Total system processing capacity

Platform performance is measured while the system is in an operating status guaranteeing the upper limit of processor use for meeting the platform's realtime characteristics and including the processing that depends on the application-specific service content.

5. Benchmark Program Configuration

5.1 Basic Approach

The CTRON benchmark program measures this performance data by using the following methodology. Each of the measurement methods uses, only the platform interfaces, without having to embed special processing inside the platform. This approach allows the compiler and other aspects of the development environment to be taken into account when comparing and evaluating the measured data. (See Figure 3.)

5.2 Measuring Fixed Overhead

The fixed overhead in a platform consists of the processing time for the cyclic scheduling mechanism that uses clock interrupts to guaranteeing realtime characteristics and of the processing for event monitoring, using cyclic look-ins for efficient processing of events from I/O devices (group processing).

(1) Data collection method

A program that does not issue system calls (Dhrystone program, a program that does not include processing for masking interrupts) is run on the Basic OS as a task. The execution time is then measured with and without cyclic processing (i.e., with clock interrupts masked and not masked). These two results are compared and used to calculate the proportion of overhead that is fixed.

(2) Measuring Object

The relation between fixed overhead and active I/O devices (i.e., the relation between fixed overhead and the types and numbers of cyclic activation) is measured separately for each system.
5.3 Measuring System Call Execution Time under Ideal Conditions

Under ideal conditions the execution time is measured for each of the functional units provided by a CTRON-conformant Basic OS. That is, the minimum execution time is measured under the parameter conditions specified for a system call, in an environment assuming that the cache/TLB hit ratio is 100 percent and that restrictions on load are outside the scope of the measurements.

(1) Data collection method

To get as many as possible of the relevant instructions and data into the cache memory (thereby minimizing the effects of cache/TLB mis-hits), a “dry run” is made of each system call before starting to measure its execution time. Then the system call to be measured is executed for a specified number of times (which can be designated in a parameter), and the average execution time is calculated.

(2) Measuring Object

Data is collected for evaluating the execution time of the system calls and other elements listed in Table 4.

5.4 Measuring System Call Execution Time under actual System Conditions

Here system calls the time required for execution of is measured under the various load conditions of actual system operation. The behavior of application programs executed in a system is simulated, and multiple tasks are made to run concurrently. In this environment, the system call execution time, processor use rate, number of concurrent tasks, and event response time are measured.

(1) Data measurement method

Application characteristics are simulated according to the types of system calls issued in a task and the number of times they are issued (can be designated in a parameter). They are also measured according to the number of times programs for application-specific processing (Dhrystone, table search, or other some program that does not issue system calls) are executed (can be designated in a parameter). Load fluctuations during data measurement are realized by changing the number of tasks created during a given length.

Figure 3. Configuration of the benchmark program.
of time (parameter designation is possible). The processor use rate and the execution times maximum, minimum, and average in this simulated environment are measured, along with the relation between load conditions and response time. Furthermore, by making task execution levels variable, the realtime characteristics of the task scheduling mechanism (including preemption) are measured as well as the relation to throughput. In addition, performance characteristics including the platform's internal algorithms are measured in relation to load characteristics.

(2) Measured data

Data for the parameters listed in Table 5 are measured while using the measurement time, number of tasks executed, their execution time, and the number of asynchronous receive operations (IO notification) as load parameters.

6. Evaluation Results

Typical result of benchmark test measurements results are used to discuss the evaluation of results.

6.1 Evaluating Fixed Overhead

To clarify each systems overhead for I/O device activation, the fixed overhead for each system was measured as illustrated in Figure 4. The difference between the performance of each platform is attributable to the different ways they monitor events. Platform implementation 2 uses a comprehensive hardware monitoring method for external I/O devices and thus achieves uniformly low overhead regardless of differences between the I/O devices activated in the system. With platform implementation 1, on the other hand, the fixed overhead values differ according to the differences in the I/O devices activated.

6.2 Evaluating System Call Execution Time

Measuring the time needed to execute an individual system call is an effective means of objectively evaluating OS implementation technology. Since the number of resources each OS functional element uses varies with the system implementation, performance values can be measured by using as a parameter the number of resources an element is issued (see Figure 5). In implementation 2, where system

| Table 4. Parameters measured under ideal conditions. (The abbreviations indicate system calls in the CTRON interface specifications.)

| Task management | CRE_TSK, STA_TSK, CAS_TSK, EXT_TSK, EXD_TSK, SUS_TSK, RSM_TSK, WUP_TSK, SLP_TSK, TER_TSK, DEL_TSK, WAL_TSK, GET_TSI, GET_TIF, GET_TID, task switching time when task is preempted |
| Message box control | CRE_MBX, DEL_MBX, SND_MSG, REM_MSG, REL_MSG, GET_BID |
| Private timer control | INI_PTM, STA_PTM, STP_PTM, RST_PTM |
| Memory management | CRE_MPL, DEL_MPL, ALC_MEM, FRE_MEM, ALC_MMS, FRE_MMS |
| Event control | GET_EVF, DEL_EVF, SET_EVF, WAL_EVF, GET_EID |

| Table 5. Measured system calls (Actual system conditions)

| Kernel system calls | CAS_TSK, SND_MSG, WUP_TSK, ALC_MMS, FRE_MMS, CRE_TSK, STA_TSK, DEL_TSK, KA_DIS_LLM, KA_ENA_LLM, GET_TSI |
| IOCS system calls | WRI_BIO (external IO device 1), WRI_BIO response time, WRI_BIO (external IO device 2 [set]), WRI_BIO (external IO device 2 [cancel]) |
calls are issued repeatedly, the average execution time increases with the number of times a system call is issued. In implementation 1, on the other hand, the average execution time is fixed. Implementation 1 uses an algorithm that is appropriate for conditions under which many resources are available, whereas implementation 2 uses an algorithm for conditions under which few resources are available.

6.3 Evaluating Basic Platform Performance

Total system performance under ideal conditions can be deduced as a scalar quantity (index \( i, j \): system model, \( j \): resource condition model), as in Equation (1) (see Appendix), based on the performance data \( (P_j) \) of platform functional elements as measured on a benchmark program and based on system call MIX \( (M_i) \) the number of times system calls are used when applications are implemented (a value determined by the characteristics of application programs in the platform application field: MIX value in other word system call frequency).

\[
\text{Index } ij = M_i \cdot P_j \quad \cdots \cdots(1)
\]

This result is used as an index of performance.

![Figure 4. Typical comparison of system fixed overheads.](image)

![Figure 5. Average time for executing a system call in each of two typical implementations.](image)
when applications are implemented. Figure 6, the results of evaluating total performance (including fixed overhead) when a system is implemented, show that platform implementation 1 is superior to implementation 2 under many multiplicities of system resource conditions (when system calls are issued frequently), whereas platform implementation 2 is superior when system load is light (when system calls are issued infrequently). When system B is implemented, platform implementation 2 offers superior performance under all load conditions.

7. Conclusion

This paper has proposed a platform evaluation method using a benchmark program. Although a benchmark program offers a simple and effective way to quantitatively measuring system performance, it does not evaluate platform performance accurately when the measurement environment of the benchmark program does not match the actual application running environment. The benchmark program proposed here simulates the running environment of realtime systems for communication processing. Moreover, within that environment it evaluates system call performance by using execution time as a performance index. It is therefore able to provide an unbiased, quantitative comparison between the performances of different platforms.

Future studies are being planned for actually creating the various benchmark programs proposed here, as well as for creating the evaluation tools for collecting and analyzing trace data, etc. These studies should enable qualitative evaluation of platform performance for realtime systems in the communication processing field.

Acknowledgments

We thank the member of CTRON technical committee and the project team members in the NTT Communication Switching Laboratories for their helpful suggestions.
Appendix

In equation (1) vector $P_j$ shows measured execution time per system call on resource condition $j$.

$$P_j = \begin{bmatrix} p_{j1} \\ p_{j2} \\ \vdots \\ p_{jk} \end{bmatrix} \quad \text{Execution time of System call 2 on resource condition}$$

$n$: Number of system call

Matrix $P$ shows execution time per system call on each resource condition

$$P = \begin{bmatrix} p_1 & p_2 & \cdots & p_k \\ \vdots & \vdots & \ddots & \vdots \\ p_1 & p_2 & \cdots & p_k \end{bmatrix} \quad \text{k: Number of resource condition variation}$$

In equation (1) vector $m_i$ shows number of occurrences per system call, showing target system i characteristics, MIX

$$m_i = \begin{bmatrix} m_{i1} \\ m_{i2} \\ \vdots \\ m_{in} \end{bmatrix} \quad \text{Number of system call occurrences on target system i}$$

Matrix $M$ shows each target system characteristics, MIX.

$$M = \begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1n} \\ m_{21} & m_{22} & \cdots & m_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{11} & m_{12} & \cdots & m_{1n} \end{bmatrix} \quad \text{l: System model}$$

Matrix $\text{Index}$ shows system performance index on each system models on each resource conditions. It is important to evaluate the platform performance.

$$\text{Index} = M \cdot P$$

System performance index

References