A Process For Performance Evaluation Of Real-Time Systems

Andrew J. Kornecki
Embry Riddle Aeronautical University, Daytona Beach, FL
kornecka@erau.edu

Eric Sorton
Command and Control Technologies Corporation, Titusville, FL
eric@cctcorp.com

ABSTRACT

Real-time developers and engineers must not only meet the system functional requirements, but also the stringent timing requirements. One of the critical decisions leading to meeting these timing requirements is the selection of an operating system under which the software will be developed and run. Although there is ample documentation on real-time systems performance and evaluation, little can be found that combines such information into an efficient process for use by developers. As the software industry moves towards clearly defined processes, creation of appropriate guidelines describing a process for performance evaluation of real-time system would greatly benefit real-time developers. This technology transition research focuses on developing such a process. PROPERT (PROcess for Performance Evaluation of Real Time systems) - the process described in this paper - is based upon established techniques for evaluating real-time systems. It organizes already existing real-time performance criteria and assessment techniques in a manner consistent with a well-formed process, based on the Personal Software Process concepts.

Keywords: Software Performance, Software Process, Software Evaluation, Real-Time Systems

1. INTRODUCTION

Real-time software must be deterministic when reacting to external events. Real-time software must not only perform in a reliable and efficient manner, within specific time constraints. If a real-time system cannot meet its specific time constraints, the software is deemed a failure. In extreme cases the failure could mean catastrophe and even loss of life.

The operating system allows the developer to focus on the purpose of the application rather than the details of interfacing with the computer. The current explosion of the Real-Time Operating Systems (RTOS) market [1] allows the developer to focus on the functionality of the application. However, the developers must rely on RTOS to provide the unique timing constraints needed to satisfy real-time system requirements. Before a system developer can use RTOS to build real-time application, an important question must be asked: which real-time kernel meets the needs of the software requirements?

One of the problems with evaluating real-time systems is that there is very limited guidance on how to measure the metrics and/or properties of the selected kernel [2]. The only viable option is to undertake a rigorous performance evaluation of selected real-time kernels to determine which best will meet the needs of the system requirements. A defined process guiding the developer through the tasks to complete the performance evaluation study, allows for the planning and tracking of the study, and provides a means to improve the process. It will allow the developer to focus more on the technical issues of performance evaluation and less on how to complete the study.

Three key areas addressed in the paper are: (i) the concept of process and how to develop a process, (ii) computer system performance evaluation focused on how to design, implement, and analyze performance of software systems, and (iii) identification of key metrics of real-time kernels and techniques which can be used to measure them. A brief description of these three topics leads into design of PROPERT (PROcess for Performance Evaluation of Real-Time operating systems). In addition, a sample performance evaluation case study is described to provide an example of how to use the PROPERT.

2. PROCESS ISSUES

A process is a partially ordered set of activities to achieve a goal [3]. The development of a process can be divided into two steps: (a) define a goal, and (b) develop a set of activities to achieve the goal. A process is a dynamic entity that is always undergoing change. The process development itself is an iterative activity with user feedback providing direction as to how the process should be modified, enhanced, and expanded.

Well-defined processes within the software industry have proven effective in increase quality, decrease cost, and improve the predictability of the software development. On the individual level, the Personal Software Process (PSP) developed by Watts Humphrey [4] guides in performing the software development tasks. The PSP helps to plan and track work being completed while helping to evaluate and improve the way the work is done. It supports repeatability and facilitates exchange of information between personnel working on the project. By planning, tracking, and recording how long it takes to complete a task, the information can be used later to analyze the work done and thus provide suggestions for ways to improve process. An extension to the PSP is Team Software Process (TSP), which is dedicated to small team project. In addition to the individual process the team roles and responsibilities are defined and the forms and scripts facilitate orderly conduct of the project [5].

The key components for a process are: scripts, forms, standards, checklists, and process improvements. Scripts describe how to perform the process task (a list of steps). Forms and process improvements are the products of the process used to record information on planning the task, the time it has taken to

complete the task. When filed they represent actual deliverables of the process. Standards and checklists contain the information related to the process that aids in generating the products and/or implementing the scripts.

3. PERFORMANCE EVALUATION ISSUES

A clear, concise statement of the performance requirements is a pre-requisite of a successful evaluation study. The requirements must be defined in a SMART way (Specific, Measurable, Acceptable, Realizable, and Thorough) [6]. The system inputs and outputs must be described from the perspective of the services the system shall perform. Typically, one divides the system into a "system under test" (SUT) and a "component under study" (CUS). The SUT is the overall system under which the services are defined. The CUS is the individual component within the system that is being examined.

Selecting the metrics is the next key step in performance evaluations. A custom built monitor, or an appropriate COTS product, are used to measure the performance metrics. The list of factors, i.e. parameters affecting the metrics of the system performance, must be compiled including all items having a negative or positive impact on the system performance. The workloads, appropriately representing the conditions of the system operations, must be selected to simulate the environment under which the system runs. Experiment design techniques [7,8]. can help alleviate some of the problems of having too many factors and/or workloads.

The execution consists of running the designed experiments changing factors under the specified workloads, while measuring performance metrics. Experiments that take place in a controlled environment can be repeated if necessary. Some of the following activities help control the environment and ensure a successful test [9]: verification of initial conditions and load generation, running standard tests to ensure correct operation, using fresh media, keeping detailed logs with time stamps, etc. It is recommended to develop a checklist including the resources needed for the experiment, the steps to be performed to complete the experiment, and the outcome of the experiment.

Execution of the performance experiments can produce large volumes of raw data. A thorough statistical analysis is required to present the results in a simple, understandable, and usable format. Statistical analysis [10] plays an important role in reducing the data and analyzing the results of the experimental data.

4. REAL-TIME SYSTEMS ISSUES

A typical example of real-time software would support an external device that needs servicing once every 10ms. On each service request, a thousand bytes of data need to be transferred. The data are read off of the interface and passed to a second process, which process the data and writes them to disk for subsequent processing. The presented example identifies services that take into account such characteristics as throughput, responsiveness, determinism, overload, and memory. Identification of these services is the key to the performance analysis. The key services, requiring deterministic performance, are: clocks and timers, scheduling and task/process management, intertask communication, interrupt response time, resource locking, signals, input/output operations, and memory management.

What are basis for performance analysis of real-time systems? The speed is not the basic feature of real time system. A real-time system is one in which the success of the system in meeting its requirements is defined not only by the functional requirements, but also by the timing constraints [11, 12]. Missing the timing deadline and unpredictability of timing are typical disqualifying features for real time system.

Selection of proper performance metrics is critical. Consider a hard real-time system which requires interrupt dispatch latency for a task of 10 microseconds. An analysis of the operating system determines that the average response is 7 microseconds with a worst case above 10 microseconds about 1% of the time, and only under heavy loading. For the hard real-time system, this behavior would be considered unacceptable. For a soft real-time system, the above behavior would most likely be acceptable. However, if the system missed the 10 microseconds deadline 90% of the time under heavy loading, the system would most likely need reexamined.

The performance evaluation of a real-time system focus on the assessment of one or more of the services listed above. The type of metric to be measured, based upon the performance evaluation requirements, are: throughput, responsiveness, and determinism [2].

Throughput is the maximum number of operations that the system can perform in a given time period. Responsiveness describes how quickly the system will respond to a particular event. Determinism indicates how predictable the system is responding to events.

The three above metrics can be illustrated by examining a single characteristic. All multi-tasking, pre-emptible real-time operating systems must have method by which to synchronize processes. One such method is to use semaphores. A metric representing throughput is the number of semaphore operations system can perform within a one second. Responsiveness is measured by determining how long it takes for a semaphore signaled by one task to wake up another task. Determinism is measured as the variance of the response time, e.g. checking if the second task is always woken at the same time, or if the time varies greatly for different semaphore calls.

Which of the three metrics is the most essential to measure? For a soft real-time system, throughput is probably the most important followed by responsiveness and then determinism. Since the timing deadlines for a soft real-time system are not as stringent, the determinism is not a critical factor. However, for a hard real-time system, in which timing deadlines are critical, determinism becomes the most important factor with responsiveness and throughput taking second place.

5. PROCESS DESCRIPTION

Based upon process literature [4], the following steps are followed in developing PROPERT:

- Determine the Priorities
- Determine the Products
- Build the Scripts
- Build the Forms
- Build the Lists/Standards

The PROPERT overall script and detailed scripts for each of the phases, as well as the forms and checklists can be found on: http://faculty.erau.edu/korn/eric/project/list-1.html .

Priorities

The process priorities describe the objectives of the process and the deliverables of the process. The following list of items reflects the performance evaluation objectives [6]. The resulting process must:

a) Produce Significant and Accurate Performance Measurements
b) Produce Accurate and Understandable Analysis of Performance Measurements
c) Minimize the Time to Produce Performance Measurements
d) Provide for Reusable Performance Techniques and Measurements
e) Accurately Predict the Time to Complete the Performance Measurements
f) Provide a Database of Performance Measurements, Metrics, Designs

The first two items address the functional aspects of completing the performance evaluation study in a manner so that it produces the desired results. The presented work focuses on these first two items.

Products

The process products are items supporting accomplishment of the priorities. The items can be categorized into three areas: records, reviews, and improvements:

Records:

1) Performance Measurement Goals/Requirements
2) Time to Complete Performance Design
3) Performance Measurement Attributes (metrics, workloads, etc ...)
4) Time to Complete Performance Implementation
5) Time to Complete Performance Analysi

Reviews:

1) Performance Measurement Design Using a Common Mistakes Checklist
2) Performance Measurement Design vs. Requirements
3) Performance Measurement Implementation Using Common Mistake Checklist
4) Performance Measurement Implementation vs. Design
5) Performance Measurement Analysis Using a Common Mistakes Checklist
6) Performance Measurement Analysis vs. Design

Improvements:

1) Methods by Which Process Improvement Can Be Performed
2) Attempt to Minimize Design/Implementation/Analyze Time Through Reuse
3) Provide Common Lists of Performance Attributes (metrics, workloads, etc ...)

Scripts

The performance evaluation study follows phases mimicking the waterfall development model with design, implementation, and analysis phases. In addition, the planning and postmortem phases are added to satisfy some pre- and post- performance evaluation activities.

The *planning script* guides the developer through the planning of the performance evaluation. The time to complete the evaluation is determined and a basic schedule is developed. The system requirements and goals are discussed, analyzed, and recorded. The *design script* guides the developer through the design of the performance evaluation. During the design phase, the system and components are analyzed and a description of both the System Under Test (SUT) and Component Under Study (CUS) are developed. In addition, the services provided by the system are described. Next, in the design phase, the metrics are selected and the monitors are described. After that, the significant factors are recorded and the workloads to be placed on the system are determined. Finally, the individual experiments are listed and discussed. A majority of the forms developed for PROPERT aid the developer in design phase.

The *implementation script* aids the developer in the implementation of the performance evaluation. This is the phase were the developer builds the actual experiments and executes them. The *analysis script*, presents some basic information on the analysis of the performance data. The analysis of the performance data and the creation of the report are the key deliverables that produced in the analysis phase. The *postmortem script* guides the developer through the final stages of the performance study. It is in this phase that the process information is completed and examined. In addition, the overall performance evaluation is examined and determined if it is acceptable or not. If it is deemed unacceptable, the process should be repeated.

Forms

The purpose of the forms is to guide the developer through the performance evaluation while recording the information in a consistent, convenient format. A total of thirteen forms were developed for the process. They can be found on line on: http://faculty.erau.edu/korn/eric/project/list-1.html (along with the instructions on how to complete them).

- Process Plan Summary (PPS)
- Process Improvement Proposal (PIP)
- Time Recording Log (TRL)
- Defect Recording Log (DRL)
- Task Planning Template (TPT)
- Schedule Planning Template (SPT)
- Statement of Goals/Requirements (SGR)
- SUT/CUS Description (SCD)
- Metric Description Form (MDF)
- Performance Test Design Form (PTD)
- Experiment Implementation Guide (EIG)
- Performance Test Data Collection Forms (PTC)
- Performance Test Data Analysis Forms (PTA)

Each of the forms, explains its purpose, and presents a brief description of the data contained within. The forms were designed using information collected and presented during the research phase of the project.

Lists/Standards

The presented defect standard describes the different possible defects that can be encountered during the performance evaluation study. It is used in conjunction with the defect recording log. By examining the defects that occur most often, the checklists described next can be customized to aid in catching and preventing the most common defects before the become costly. The defect standard was developed based upon research of real time system performance.

Three checklists were developed: *Design, Analysis*, and *Implementation* Checklists. The checklists are questions to ask

after the specific phases to determine if all of the important criteria have been met. The purpose of the checklist is to help eliminate defects from the performance evaluation before they become too costly.

6. CASE STUDY

Modern POSIX compliant UNIX systems allow the user to create a file on the disk drive and subsequently memory map that file so that it can be accessed as if it were a "conventional" memory [13]. The memory map returns a virtual address to the file which can be used in C programs for such operations as assignment, comparison, copy, etc... In addition, most modern UNIX systems as well as the hard disks themselves have sophisticated caching systems, which can greatly enhance the performance of the memory mapped file implementation.

One question is whether or not is there a valid use for this scheme. Two immediate advantages come to mind. First, hard disks are rather inexpensive. If an application calls for a large amount of memory, but does not necessarily need to access that memory at high speed (e.g. for archiving), memory mapping of a file might be the acceptable solution. A large file, of the order of several GBytes, could be memory mapped and treated as main memory. An additional advantage is that the memory-mapped file provides for a permanent record of the state of the system (up to the last cache write) in the event of a crash. A second question that should be addressed is why not just use traditional file access methods such as *open(), read(), write(),* and *close()*. If the purpose of an application is to read and write sequential data to and from the disk, reads/writes are probably the correct choice. However, for random access or non-sequential read/write, the traditional model becomes bulky and difficult. The developer must seek the location on the disk, read the data, seek a different location, and write the data, etc. This can quickly become cumbersome. By memory mapping the data and overlaying the memory area with program data structures, the compiler will determine the offsets automatically, which will greatly simplify the programming tasks.

To determine performance of *mmap()*ed files the following objectives were identified:

- Define the System Under Test (SUT) and Component Under Study (CUS) - Table 1
- Define all relevant inputs, outputs, and factors affecting the performance of *mmap()*ed files - Table 2
- Determine a set of metrics that can be used to measure describe the performance of *mmap()*ed files - Table 3
- Identify the monitors to be used to capture the defined metrics - Table 4
- Identify the workloads - Table 5
- Develop a set of experiments to measure the determined metrics - Table 6
- Analyze the results of the experiments

7. CONCLUSIONS

The described case study created PROPERT deliverables. The deliverables proved very useful to organize the development of performance evaluation study. During the validation tests, PROPERT helped in the completion of the performance evaluations by reducing the effort involved in study preparation and execution. Several key process improvements were a direct result of this verification and validation test.

Three areas were identified for further improvement. These areas include expanding the experiment design, data processing and presentation, and independent validation. Additional implementation by other developers is needed to verify the process usefulness. Independent validation simply consists of using the process and assessing whether or not it actually accomplishes the original goals. Such independent test is an important step in maturing the process. The presented evaluation process could be further enhanced by process improvements. After each use of the process, the users can make additional refinements and enhancements. This is expected since process improvement is an integral part of the process. There are other areas of the process that are weak due to a lack of valid statistics. As the process is used, the most common defects (as recorded in the defect recording log) should be identified. The checklists can then be modified to help catch these defects at their inception and possibly before they occur.

PROPERT shall aid developers who need to analyze a real-time system to determine whether or not it meets specific performance goals. It should simplify design of the performance evaluation study, produce a better product, and reduce the time necessary to complete the evaluation. With the strong trend towards process-based development, we believe that PROPERT artifacts will be found useful in the software engineering community.

8. ACKNOWLEDGEMENT

The authors would like to acknowledge the contribution of Rick Bard to the presented memory-mapping case study.

9. REFERENCES

[1] M. Timmerman, The RTOS Buyer's Guide, Dedicated Systems Encyclopedia http://www.dedicated-systems.com/encyc/buyersguide/rtos/Dir228.html
[2] B. Gallmeister "POSIX.4: Programming for the Real World" O'Reilly & Associates, 1995
[3] P. Garg, M. Jazayeri "Process-Centered Software Engineering Environments." IEEE Computer Society Press, 1996
[4] W. Humphrey "A Discipline for Software Engineering" Addison-Wesley, 1995
[5] W. Humphrey "Introduction to Team Software Process", Addison-Wesley, 2000
[6] R. Jain "The Art of Computer Systems Performance Analysis, Techniques for Experimental Design, Measurement, Simulation, and Modeling" John Wiley & Sons, 1991
[7] R.L Masone, R.F. Cunst, J.L.Hess "Statistical Design and Analysis of Experiment" Wiley, 1989
[8] D.C. Montgomery "Design and Analysis of Experiments" Wiley, 1984.
[9] B. Beizer "Software System Testing and Quality Assurance", Van Nostrand Reinhold Company, 1984.
[10] G.E. Box, W. Hunter, J.S. Hunter "Statistics for Experimenters" Wiley, 1978.
[11] A. Burns, A.Wellings "Real-Time Systems and Programming Languages" 2nd. Ed. Addison-Wesley, 1996
[12] B. Furht, D. Grostick, D. Gluch, G. Rabbat, J. Parker, and M. McRoberts "Real-Time UNIX Systems: Design and Application Guide" Kluwer Academic Publishers, 1991.
[13] T. Schieber."Sharing Memory with Memory-Mapped Files" Unix Review, October 1997, pp 47-53.

10. SAMPLE FORMS

Table 1: SUT/CUS

System Under Test	Description
UNIX OS w/ Memory Map Capablities	POSIX compliant OS have the ability to map a file into virtual memory and therefore access the file using standard C memory operations such as assignment, comparison, and *memcpy()*. Any operating system with the _POSIX_SHARED_MEMORY_OBJECTS and _POSIX_MAPPED_FILES defined in the *unistd.h* supports memory mapped objects. In addition, to support this testing, the operating system should have _POSIX_PRIORITY_SCHEDULING and _POSIX_MEMLOCK defined which support the testing by changing the priority of the test programs and locking the test programs into memory. For the purpose of these tests, the SUT should be at a minimum a dual processor machine. One processor is needed to run the test suite and the second processor is needed to load the disk. In these experiments, a Silicon Graphics Octane with two R10000 processors is used.
Component Under Study	**Description**
Memory Map Facilities	The actual component under test is the specific memory map facilities of the operating system and the underlying components that allow the memory mapped files to work correctly. The memory map facilities of the POSIX operating system consists of *mmap(), munmap(), ftruncate()*, and *msync()* function calls. Only the *msync()* call will be measured directly. The other facilities will be used to memory map the file and then accesses into the virtual memory will be analyzed.

Table 2: Inputs/Outputs/Factors

Inputs	Description
Map Request	A request to map a region of memory, the *mmap()* call, is an input. The will not be measured directly.
Write	The primary input into the memory-mapped region is a write. This can take form of an assignment statement (lhv) or a C function call such as *memcpy(), strcpy(),* or others.
Sync	A request to synchronize the memory-mapped region with the underlying hardware can also be treated as an input to the system. The *msync()* call takes several options and informs the OS to perform several different functions.
Un-map Request	A request to un-map a region of memory, the *munmap()* call, is an input. The will not be measured directly.
Outputs	**Description**
Read	A read is the output of the memory mapped region. This can take many forms from a assignment statement (rhv), a comparison, or a C call such as *memcmp(), strcmp*(), or others.
Factors	**Description**
Disk Activity	The current state of the disk activity is an attribute that can affect the timing of reading and writing from and to the memory mapped region. Theoretically, the busier the disk, the slower the access will be.
Access Size	The access size is attribute that can have an affect on the performance of the system. Smaller access sizes forces the overhead and thus degrades performance. Larger access sizes allow the OS less overheads by grouping operations together.
Access Order	The order in which the data is accessed could affect the performance. Sequential access with read-ahead caching could improve performance. Random access will invalidate the read- ahead cache and could affect performance in a negative way.
Caching	The state of the cache could also affect the system. The state of the cache can be changed by the parameters given to the *open()* system call and the use of the *msync()* flag.

Table 3: Metrics

Num	Metric	Type	Description
M1	Latency Time	Responsiveness	The latency time will be measured to determine how long it takes for the system to propgate a write to other processes in the system.
M2	Bytes/Sec	Throughput	The number of bytes per second will be measured to determine the throughput of the system. This metric will primarily be used to see how quickly reads/writes occur.
M3	Time per Operation	Throughput	The time to complete an operation or the number of operations per second will be used to determine the performance of several of the calls used in the experiments.
M4	Disk Activity	Throughput	The disk activity will be measured since it is an important factor that affects performance.

Table 4: Monitors

Num	Monitor	Type	Monitor Description	Metric List	Intrusion
O1	Experiments	Custom	M1, M2, M3	The experimental applications written to support the testing	Low
O2	System Monitor	COTS	M4	Utilities available to measure disk activity. This will be used to monitor disk activity.	Low

Table 5: Workloads

Num	Workload Description	Constant Factors/Services	Varying Factors/Services
W1	Idle System	---	---
W2	Synchronization	All other factors/attributes are constant	With/without *msync()*
W3	Disk Loading	All other factors/attributes are constant	Disk activity is varied
W4	Cache Status	All other factors/attributes are constant	*open()* with/without O_SYNC
W5	Block Size	All other factors/attributes are constant	Vary the block size

Table 6: Experiments

Num	Experiment	Experiment Description	Metric List	Workload List	Num of Runs	Run Description
E1	Time	Measure the time it takes to call the time measurement function.	M3	W1	250	Each run is identical. Find Avg, Min, and Max. The algorithm to collect the clock data is simple loop, retrieving the clock data each time. By checking the difference in the time retrieved, the time it takes to call the clock function can be determined.
E2	Sync	Measure the overhead associated with a *msync()*.	M3	W1	250	Each run is identical. Avg, Min, and Max will be found.
E3	Latency	Measure the round-trip latency of reading/writing data.	M1, M4	W2, W3	250X4	2^k factorial design with four runs of 250 times each will be performed. Avg, Min, and Max will be found. Synchronization and disk access will vary.
E4	Throughput	Measure the round-trip latency and rate of large data blocks.	M1, M2, M4	W2, W3, W4, W5	250x16	2^k factorial design of sixteen runs of 250 times each will be performed to find Avg, Min, and Max.. Synchronization, disk access, cache status, and block size will vary.

Example results for one of the metrics (M1 - responsiveness) are presented in Table 7. Typical graph reflecting the collected data for one of the experiments is shown in Fig 1.

Table 7: Results-Responsiveness

Responsiveness Results
Examining the results shown in the line chart graphs generated from the raw data, the following information can be extracted: (1) When using MSYNC, the latency - after removing the obvious outliers - is approximately 15 msec. This is true regardless of the state of the disk loader. (2) When MSYNC is not active, the latency is approximately .17 msec, regardless of the state of the disk loader. Using information from the LATENCY 2^2 Sign Table (1) The mean latency performance of the system is approximately 32 milliseconds. (2) The effect of the MSYNC is 32 milliseconds, and the effect of the disk loader is 2.9 milliseconds (3) Using the **Sum of Squares Total (SST)** information, the disk loader is responsible for over 34% of the variation, while the MSYNC is responsible for 11.7% of the variation. **COMMENTS:** These results appear to be contradictory. Commonly held knowledge indicates that the **SST** data should be true: I/O is the most expensive activity that can be performed on a machine. In the line chart graphs, the DISK-LOADER environment should have produced results that were similar to that of MSYNC, because both DISK-LOADER and MSYNC can be viewed as I/O activities. The results were obtained on a 2-processor SGI Octane machine with 256 MB memory and R10K processors. These results are different from those obtained on a PC running LINUX. In a less time-constrained environment, the next step would be to use performance analysis monitoring tools to verify that the desired behavior is in fact the behavior that is taking place. It has been validated that the disk loader is indeed producing a file of the anticipated size during these test runs.

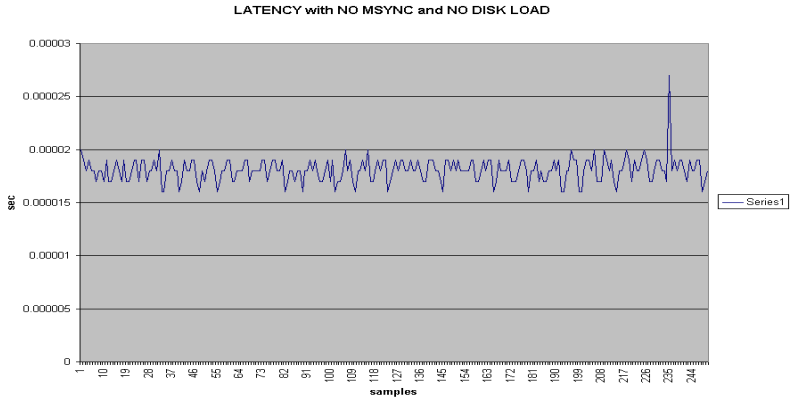

Figure 1