

Design and Implementation of Performance Counters for Real Time Database Server Clients

Asia Ali Salman* 

Received on: 5/1/2009

Accepted on: 16/2/2010

Abstract

The aim of this research is to monitor, check, and do the necessary refinements for the performance of clients working with Database server in the real time environment. If there are a number of clients in the network need to access Database server designed for distributed real time system in particular sequence, this will cause many bottlenecks in the system, making the work of the system unstable, especially in critical systems such as that of power and water distribution. A designed performance counters and objects were added to each client to know who makes the bottlenecks, also they will be used while developing and debugging the clients when they access the Database server on the network in order to tune the performance of the system. After completing the designed system and installing it at the target, the counters can help system administrators to adjust configurable settings for that system. Using this new performance counters in the execution time will help to see the effect of clients on each other, on network, and on performance of the Database server. The results show that, the designed performance counters can detect the bottlenecks which are caused by week points in Client's program code, so they will help the programmers to amendment and redistribute the client's program code on the network with minimum errors when accessing DB server. Windows NT/XP/2000 provides a mechanism for developers to add performance objects and counters for their applications and other software components. These objects and counters can provide performance data to Windows NT/XP/2000 Performance Monitor.

تصميم وتنفيذ عدادات الأداء لزبائن خادم قاعدة بيانات أنظمة الوقت الحقيقي الخلاصة

الهدف من هذا البحث هو مراقبة وفحص وعمل التحسينات الضرورية لأداء الزبائن العاملين على خادم قاعدة بيانات في محيط الزمن الحقيقي. إذا كان هناك عدد من الزبائن داخل الشبكة بحاجة للوصول إلى خادم لقاعدة بيانات صمم لأنظمة الزمن الحقيقي الموزعة ويتسلسل معين فإن ذلك سوف يسبب عددا من الاختناقات في النظام مما يجعل العمل عليه غير مستقر خاصة في الأنظمة الحرجة مثل أنظمة توزيع الطاقة و الماء. قمنا بتصميم عدادات وأهداف (objects) الأداء لمعرفة أي زبون هو المسئول عن الاختناقات , وأيضا للاستفادة منها أثناء عملية تطوير وتصحيح الأخطاء للزبائن أثناء وصولهم إلى قاعدة بيانات خادم الشبكة من أجل تنظيم أداء النظام. وبعد الانتهاء من تصميم النظام وتنصيبه على حاسبات الهدف (المنظومة)، العدادات المضافة سوف تساعد مدراء النظام على تعديل تنسيق الزبائن على حاسبات النظام. إن استخدام عدادات الأداء الجديدة أثناء وقت التنفيذ سيساعد على رؤية تأثير الزبائن بعضها على بعض وعلى الشبكة وعلى أداء خادم قاعدة البيانات. النتائج أثبتت انه سيكون بالإمكان الكشف عن الاختناقات الحاصلة بسبب النقاط الضعيفة في شفرة برنامج الزبون فتستطيع المدراء مرمج على أعـدادة تحسين وتوزيع شفرة برنامج الزبون على الشبكة وبالحد الأدنى من الأخطاء عند وصوله لخادم قاعدة البيانات.

أنظمة التشغيل (NT/XP/2000) وفرت ميكانيكية لمطوري الأنظمة لإضافة عدادات وأهداف الأداء لتطبيقاتهم و لمكونات البرامج. عدادات وأهداف الأداء تقوم بتزويد بيانات الأداء لمراقب الأداء.

1. Introduction

Real time applications are among the most demanding of applications. These applications not only need to respond correctly; they also need to respond within certain specified time parameters, or in "real time". Examples of these applications include (Manufacturing process controls, Medical monitoring equipment, Telecommunications switching equipment, High-speed data acquisition devices) [1].

Real time applications need high attention to the time it takes to start, to work in accurate ways, and without stopping. Therefore using performance counter for critical task in order to help in speed development time, add and validate new features. They are implemented using software variables or hardware registers; an example of a performance counter is one that counts the throughput of bytes to and from a server application [2], this paper uses a method which helps in adding new features to the clients, tunes there performance, and to specify which client makes access to the database server difficult from other clients.

Performance counters, it is one of many indicators (such as debugging, trussing, logging), help to determine the overall performance of an application and can also help to diagnose those undocumented features (errors or bugs), performance counter features can monitor physical system components such as processors, memory, processes, and networks, and if they are used within an application, they can capture and publish performance-related data about that application. The published

counter information is captured and then compared it against acceptable performance criteria [3].

In this research, there are a number of clients module work with the DB server, which is (part of system control and data Acquisition systems (SCADA)), according to race condition which is a situation, where several processes (clients) access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place [4], in this research performance counter will collect data at runtime about performance of all the DB clients which means a counter has been added in each client module, so when the client access the DB server to update the value of the counter, each client enter critical section.

The idea behind critical section is that; each thread that requires exclusive access to a resource can lock a critical section before accessing that resource and unlock it when the access is complete [5], then the data in the counters is accessible through the Performance Monitor. Performance Monitor also allows modifying the interval over which data is collected.

When a number of clients need access to the DB server, a performance counter starts working using a timer in order to compute the number of times each client access the DB server and which one makes the access through the network to the DB server from the rest of clients very low, then use the output of counter in performance monitor to decide how to arrange the modules on

the network and how to make the client that makes the bottlenecks more stable in its access to the DB server. The Designed system implemented using visual C++ ver 6.0 under Win NT/XP operating system.

Stijn Eyerman, et. al., [6] proposed a novel way of architecting hardware performance counters for building accurate Cycles per Instruction (CPI) Components, also Pavel Simakov components [7], illustrated the use of Linguine Watch(open-source Java library for real-time monitoring) for adding real-time monitoring to an existing Java Graph application using performance counter and SNMP support to the application, while Lee, K.-J.; et. al., [8] described a software solution for temperature sensing that uses real hardware resources such as performance counters.

2- Basic of Performance Counters

Performance counters are grouped into categories. Existing Microsoft Windows categories include Processor, processes, and Memory. The Processor category, for example, contains all of the counters related to the measuring of the performance of the system's processor. An instance is an optional further grouping of counters. Instances track data on multiple occurrences of an object represent the category. For example, a system with multiple processors would have an instance for each processor and another instance represented the total performance of all processors. That would enable us to monitor the performance behavior of any individual processor or all of them combined. Individual counters can be defined to measure whatever criteria

we desire [3].

3- Performance Counter Types

A number of different performance counter types are available. They range from items that contain counts, to those that perform calculations such as averages, [3]. The following are counter types:

1-AverageTimer32—a counter that measures the average time required to complete a process or operation. It is calculated by a ratio of total time elapsed to the number of items completed during that time.

2-AverageBase—a supporting base counter that contains the number of items completed during the sample interval.

3-NumberOfItems32—an instance counter that tracks a count of items. Could be used to monitor the number of times an operation is performed.

4-RateOfCountsPerSecond32 — a counter that tracks the number of items or operations per second.

Note: this research design three counters two of them are from the same type, which is number four above (our first counter is *RateOfCountsPerSecond32*, and the second one is *RateOfCountsPerHour*), the third counter is the same as number three above which is *NumberOfItems32* the research will describe them later

4- How to Add Performance Counters

To add performance counters to the system, an extended object (this one is performance DLL application which will be explained later) must be created, extended object called when Performance Monitor collects data, as shown in figure (1), the configuration registry

is handled by ADVAPI32.DLL in the Performance Monitor process on the local computer, and by SCREG.EXE on remotely monitored computers [9]. To monitor the clients who are working on other computers on the network using the program SCREG.EXE.

The following steps are used to design performance objects and counters for clients of the real time DB server.

Set up the necessary performance monitoring entries in the Registry. This includes the following steps [9]:

- Create a Performance key in the application's Services node in the Registry. If there is no such node then creates one under HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services.
- Create an .INI file containing the names and descriptions of the counter objects and counters.
- Create an .H file containing the relative offsets at which the counter objects and counters will be installed in the Registry.
- Use the **lodctr** utility with the .INI and .H files to install the information in the Registry. **Lodctr** succeeds only if a Performance key exists in the application's Services node.
- Add Library, Open, Collect, and Close value entries to the application's Services node in the Registry. These entries specify the name of the application's performance DLL, and the names of the DLLs required functions. The Open and Close entries are optional.

Create a performance DLL containing a set of exported functions that provide the link

between the clients and a performance monitoring application (such as Windows XP Performance Monitor).

Note: functions and data structures were added for each client in order to collect and store performance data, also the clients are provided with mechanism for communicating the data to the performance DLL, as it will be explained later.

5- Object and Counter Design

A performance counter defines the type of data that is available for a particular type of counter object [9].

Using the Windows XP/NT system to illustrate the relationship between objects and counters, objects include memory, disk, and cache. Each of these objects has multiple counters relating to that object: for example, the Memory object includes the counters Available Bytes and Page Faults/sec [9]. In this research, designing an object specific for the clients will be the name of the performance DLL that is responsible for connecting the clients DLL which containing the counters functions and data with the performance monitor as shown in the following figure (2).

Note: in Figure (2) the sequence of calls (Client DLL calls performance DLL) until reaching the Performance Monitor were done by this research. Connecting the DLL(s) with performance monitor is done by the putting them in windows\system32 directory.

Note: Performance Monitor always shows counters denoting raw counts as rates such as Page Faults/sec. This gives context to the viewer, who doesn't have to do in-the-head calculations to compare data from

different time intervals. Programmers or developers don't have to worry about this when design a counter just let the counter count incrementally, and the monitor application do the work of converting raw counts to a rate [9]. This is the way used in achieving this research when adding counters to the clients.

The following steps are used for adding entries to the registry in order to enable performance counters to the clients:

1. Create a Performance key in the application's Services node, and add value entries under it specifying the name of the application's performance DLL and the names of the DLL functions.

Note: To add necessary values in the registry database, a file with extension (.reg) has been opened in notepad program as in figure (4) with the information; library, open, collect, and close functions. The First Counter and all the three items below that are shown in figure (3) will be added in step 2 below.

- 2- Use the **lodctr** utility to install counter names and descriptions into the registry.

The **lodctr** utility also adds the value entries(*firstcounter*, *FirstHelp*,*LastCounter* ,and *LastHelp*) to the Performance subkey in the application's **Services** node, after adding the entries to the registry in the first step, then the command-line syntax for **lodctr** is:**lodctr MyApplication.ini**,[9].

6- Removing Counter Names and Descriptions from the Registry

In order to remove counter names and explain text from the Registry, which means that there is no need for performance DLL or the

performance counter, the **unlodctr** utility is used. This utility removes the Registry entries made by**lodctr**.

Use the syntax

Unlodctr is:

unlodctr *ApplicationName*

The **unlodctr** utility looks up the First Counter and Last Counter values in the application's **Performance** key to determine the indexes of the counter objects to remove. Using these indexes, makes the following changes to the Last Counter, Last Help, Counters, and Help values under the **Perflib** node, [9].

7- Description of the Performance DLL

Performance DLL defines the counter and object data structures that it is used to pass performance data to the performance monitor application. The DLL also provides up to three exported functions—Open, Collect, and Close as shown in figure (3), they are called by the Registry controller in response to requests from a performance monitoring program. The Collect function is required; while Open and Close are optional are exported in .DEF file [9].

First of all defining client performance counters and designing a counter structure is needed in order to connect to the performance monitor. More details description of the performance DLL functions are shown in algorithm (1), algorithm (2), and algorithm (3).

8- Description of Client Performance DLL

In this dynamic link library (DLL) first initialized the interface of the performance counters DLL by opening the Shared Memory file used to communicate the client with the counter DLL. If the Shared memory

file doesn't open, it will be created. If the file has already been created then it will start working by mapping the structure of the counters and initializing them. Two functions are added to the client performance DLL in order to work on, first one is the second counter function and the second one, is the hour counter function, while bytes counter is accumulated in the second function, as described in algorithm (4).

9- Description of client Module

A client module which is also called workstation or client machine computer accesses shared network resources (DB) provided by a master computer (called a server). A client application may reside on a server or on a user's computer [1]. First the client need to initialize access to the database server and connect to the server through the interprocess communication (IPC) technique using named pipes technique. When the connection established the client will tried to access the database server, in each access to the database functions (get record, set record, update record, etc) the client used a performance counter. To implement performance counter correctly, each modification of a counter value will be done inside a critical section, critical section is necessary in order to exclude the access so other client can not reach there critical section. The performance counter controlled by a windows timer see algorithm(5), windows timer is a useful programming element that sometimes makes multithreaded programming unnecessary, (if there is a need to read a communication buffer, for example, set up a timer to retrieve the accumulated characters every 100 milliseconds); also using a timer to

control animation because the timer is independent of CPU clock speed [10]. Figure (9) shows the implementation of client module in order to deal with DB server.

Note(1): in this research sending and receiving information between database server and clients are done using the named pipes technique which are a simple interprocess communication (IPC) mechanism included in Microsoft Windows NT, Windows 2000, Windows XP (in our work), Windows 95, and Windows 98 (but not Windows CE). Mailslot technique also used to establish the connection and to do some preparing situation at the beginning, and then the named pipes are work on keeping the connection between database server and its clients.

Named pipes provide reliable one-way and two-way data communications among processes on the same computer or among processes on different computers across a network. One of the best reasons for using named pipes as a networking communication solution is that they take advantage of security features built into Windows NT and Windows 2000, Windows XP (in our work) [10].

Note(2): In order to access the exported functions (call_counter_sec, and call_counter_hour) in the client performance DLL a .DEF file must be added to the client application, this file contains the library DLL name ,exported functions names, and the header file which contains the identifications for them.


10- Database Server Computer

In order to allow the clients to access the DB, a server was build to controls the access of the clients to the DB, as shown in figure (7). At

first, the server is responsible for checking the existence of the database files folder then check if there are any clients try to connect to the server in order to access its database, if any client exists it will connect to the server through the connection method (named pipes) which have several features that make them an appropriate general purpose mechanism for implementing IPC-based application. The flowchart in figure (8) shows how the DB server works with clients modules.

Figure (10) shows how DB server started successfully. Figure (11) lists the connected clients to the DB server. If the operator stops the clients, the server will disconnect them as in figure (12), while figure (13) shows that; if the server is stopped, the clients will disconnect from it since DB server no longer exists. To return the DB server to the service; click the Run button.

11- The Interaction between Performance Monitor and Client Module

When the DB server starts working and the clients begin the connection to the DB server, the designed performance counters will be activated according to algorithm (5) to specify which client makes the bottlenecks and send the collected data to the Performance Monitor. Performance Monitor could be found under *Administrative Tools* in the *Control panel*. Double-click it and a window will show up with a Y-Axis ranging from 0 to 100 and an imaginary X-Axis, which depicts the time elapsed. To add counters using the  button. In the opening window, the category *Processor* and in the category the counter *Processor time*

pre-selected by default. The categories can be changed using the dropdown listbox and the counters with the listbox under that. On the right hand side, the instances are available [11]. As shown in figure (14) removing the default counters should be done before opening the form (Add Counter) to add the new custom counters. First, chose the computer name to allow the user to indicate the computer from which performance information should be collected, and then chose the performance Object field which is a component in the system that wants to offer performance information by exposing objects.

Now, the following figures will explain the idea of this research by explaining how each client will affect on the access of each other to the DB server before and after amendment using the designed performance counters that added to all clients in the system.

figure (15) shows (client_a) before amendment its code and how its effect on the performance of the system after (10~14) second, keeping its peak position as long as it's work, and take the resource (DB) exclusively.

As mentioned before, this research work on how the new performance counters will monitor the performance of each client and help the programmer to tune the client without affecting the efficient way that works within.

Figure (16) shows the performance of (client_a) after amendment its code program through re-arranging the formula of its functions making it more stable and not stay in the peak of the graph for along time, this help other clients to

access the DB server without any bottlenecks. The figure also shows (client_a) reaching to peak of graph in the format (Minutes: Second) just between (1:20~1:28). Then figure (17) shows (client_a) which no longer keeps its access to the database at peak, actually after (35~45) Second it reaches to (60~70) % as an average of its access to the database and still in that range which is better than in figure (15).

Figure (18) shows the performance counters of (client_a) with a number of clients before amendment its code, (client_a) is always taking the DB more than any other clients, then (client_D) takes place in the second level, while client B and C seem to be with minimum access to the DB because of domination of (client_a) and (client_D) on DB server. Figure (19) shows (client_D) begins to be more stable than (client_a), also figure (20) shows the system is more stable except (client_a), a histogram in figure (21) gives a better view of what happened in the system. Figures (22,23) give bytes counters and second counter for all four clients.

The following figures show the behavior of the system clients after amendment the code of (client_a).

In figure (24) the clients are more stable and access the DB without difficulties, also figure (25) shows (client_a) starts to deal with DB after (1:20~1:28), it's no longer keep its access to the database at peak. Figure (26) shows that, after (35~45) Second (client_a) reaches to (60~70) as an average of its access to the database and keeps its access in this range which is better in performance than in figure

(15). While figure (27) shows a histogram in which the clients access the DB server in more acceptable way than in figure (21). Figures (28, 29) give bytes and second counters for all four clients after amendment.

Figures (30, 31) show the Hours Counter for the clients (A, B, C, and D). Figure (30) shows (client_a) and (client_B) hours counters which is started before (client_C) and (client_D). Figure (31) shows all four clients hour's counters.

The Results:

The results show that, the designed performance object and counters solve the problems caused by performance of clients accessing DB server which is a part of (SCADA) system. They detect the bottlenecks which are caused by weak points in Client's program code, help to amendment and redistribute the client's program code on the network with minimum errors when accessing DB, so the clients will access the DB without difficulties.

Conclusions

It's obvious that the designed performance objects and counters are an analytical method to the performance of systems in real time environment.

The values that produced through this analytical method will be monitored on *performance monitor*, so, these counters proved that they help the programmers to analyze and refine any system work on database server, also they have approved that they are effective way to discover the bottlenecks in the system and have the ability to fix it.

References:

[1]-Microsoft Corporation," real-time systems and Microsoft Windows

NT", backgrounds, windows platform MSDN Library-(2000, 2001).

[2]- Intel Corporation, "Writing Performance DLLs for the VTune™ Performance Analyzer", 2000.

[3]- Mark Strawmyer, " Performance Counters Determine Application Performance ", Jupitermedia Corporation, www\http\developer.com, 2008.

[4]-Abraham Silberschatz, Peter Baer, Greg Gagne, "Operating System Concepts", John Wiley and sons,2003.

[5]- Jeff Prosise," Programming Windows with MFC", Microsoft press A Division of Microsoft Corporation One Microsoft Way Redmond, Washington 98052-6399, 1999.

[6]-Stijn Eyerman, Lieven Eeckhout, Tejas Karkhanis , James E. Smith , "A Performance Counter Architecture for Computing Accurate CPI Components", ACM SIGOPS Operating systems review.vol 40, iss, 5, (dec. 2006).proceeding of the 2006 ASPLOS conference. pp. (175-184).

[7]- Pavel Simakov,"Adding Performance Counters and SNMP (Simple Network Management Protocol) Support to Graph Application", <http://www.softwaresecretweapons.com>, 2006.

[8]- Lee, K.-J.; Skadron, K,"Using performance counters for runtime temperature sensing in high-performance processors", Parallel and Distributed Processing Symposium,

2005. Proceedings. 19th IEEE International Volume, Issue, 4-8 April 2005 Page(s): 8 pp. - Digital Object Identifier 10.1109/IPDPS.2005.448

[9]- Microsoft Corporation, MSDN Library (2000, 2001),"windows NT Resource Kit Volume 4: How To Optimize Windows NT".

[10]-Anthony Jones and Jim Ohlund, "Network Programming for Microsoft Windows", Microsoft Enterprise Learning Library, Developer Edition 1999.

[11]-Michael Groeger,"An Introduction to Performance Counters",<http://www.codeproject.com/KB/dotnet/perfcounter.aspx> 2004.

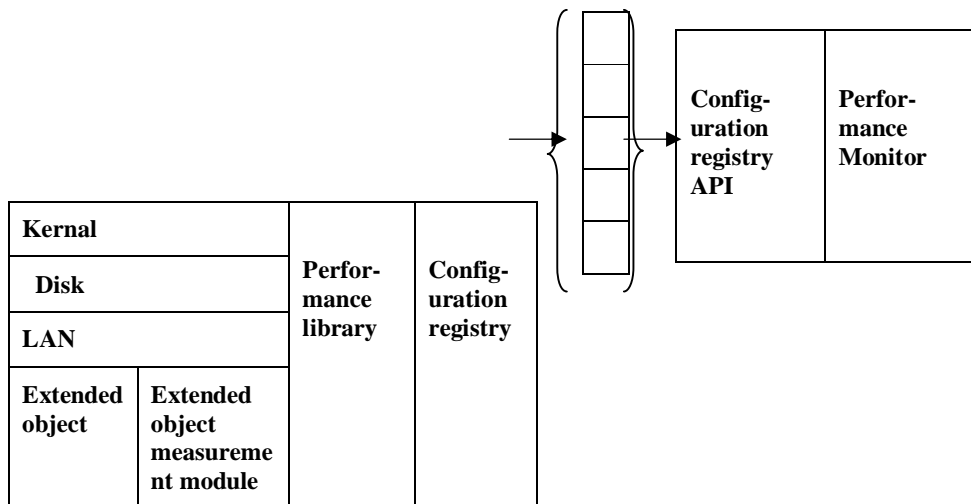


Figure (1) – Show how Performance Monitor collects data from an extended object

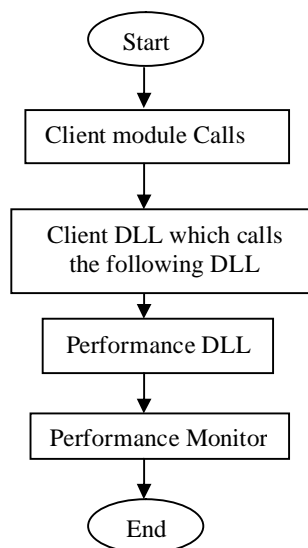


Figure (2) The sequence of calls to reach the performance monitor

```

\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\ApplicationName
(performance DLL name)\Performance
Library = DLL_Name(ex: suggested Performance DLL name)
Open = Open_Function_Name (ex: Open_Performance)
Collect = Collect_Function_Name (ex:Collect_Performance)
Close = Close_Function_Name (ex: Close_Performance)
First Counter =
First Help =
Last Counter =
Last Help =

```

Figure (3)-The code shows the values under service key

```

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\AppMem\Performan
ce]
"Library" = "appcounter.dll"
"Open" = "OpenAppcounterPerformanceData"
"Collect" = "CollectAppcounterPerformanceData"

```

Figure (4) - the Registry values

```

\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib\
009
Last Counter =
Last Help =
ex:
Counters = 2 System 4 Memory (note we will see this when we access perflib)(Note:at the end of
this text there is our counter because the system add any new counter to the end of the existing
system and programmer counters)
Help = 3 The System object type includes...
\other supported languages

```

Figure (5)-The registry location where performance counter names and descriptions are stored

```

[info]
applicationname=ApplicationName(ex:performanceDLL)
symbolfile=SymbolFile(performance.h)

[languages] // one key (value optional) for each language supported
langid=
.
.
[text] // name and description for each counter or counter object
offset_langid_NAME=Name // "Counters" name string
offset_langid_HELP=Description // "Help" description string
.

```

Figure (6) - The format of .INI file

Table (1) shows the meaning of data in .ini file (LodCtr Variables)

Variable	Description
Application Name	The name of the application found under the currentcontrolset\service key
Symbol file	An .H files containing symbolic offsets of counters. The performance DLL also uses the offsets in this file along with the First Counter and First Help Registry values to determine the indexes of the various counters and counter objects.
langid	An ID corresponding to the language subkey in the Registry (for example, 009 for U.S. English).
offset	A symbolic constant defined in <i>Symbol File</i> . Offsets must be consecutive, even numbers beginning with zero. These offsets determine the order in which the counters are installed in the Counters and Help values in the Registry.

```

1- Input: name of shared memory (that used by client performanceDLL)
2- Output: success flag
3-Open and map the shared memory which contains the performance data.
4- Initializes the data structures used to pass data back to the registry, counter structure contains three types of counters which are
(NumberOfItems32,RateOfCountsPerSecond32, RateOfCountsPerHour)
5-get counter and help index base values from registry system database (Using Applications Programming Interface (API)
functions)
    a- Open key to registry entry to read the data added to registry as in figure (3)
    b- Read First Counter and First Help values.
6-The initialization (open function) now will update static data (counter structures, which are defined by the programmer and
contained the counters definitions)
Counterstructure. NumberOfItems32.CounterNameTitleIndex += dwFirstCounter
Counterstructure. RateOfCountsPerSecond32.CounterNameTitleIndex += dwFirstCounter
Counterstructure. RateOfCountsPerHour.CounterNameTitleIndex += dwFirstCounter

Counterstructure. NumberOfItems32. CounterHelpTitleIndex+= dwFirstHelp
Counterstructure. RateOfCountsPerSecond32. CounterHelpTitleIndex+= dwFirstHelp
Counterstructure. RateOfCountsPerHour. CounterHelpTitleIndex+= dwFirstHelp
7- Close registry key (API Function)
8- END

```

Algorithm (1) - Open Function in Performance DLL

```

This Algorithm will return the data for the Application performance counters
1-Input: a- buffer contains counters names taken from registry database , b- size of buffer
2-Output: Error Flag if buffer that passed is too small to hold data
    Else Success Flag
3-Check if the open function in Algorithm (1) works correctly
4-IF (handle (pointer of Shared Memory which is opened in Algorithm (1)) =NULL) THEN
Open function Error GOTO STEP 8.
5-ELSE there is a handle (pointer) to the shared memory then copy the object & counter definition
information.
6-Process instances in the shared memory buffer which have the following values of counters:
Counterstructure. NumberOfItems32= NumberOfItems32 (the value exist in the shared buffer);
Counterstructure. RateOfCountsPerSecond32= RateOfCountsPerSecond32 (the value exist in the shared
buffer);
Counterstructure. RateOfCountsPerHour = RateOfCountsPerHour (the value exist in the shared buffer);
7- Free shared memory (call API Function)
8- END

```

Algorithm (2) - Collect Function in Performance DLL

```

1- Input: none
2-output: Success flag
3-This function close the open handle (pointer) to shared Memory
4- END

```

Algorithm (3) - Close Function in Performance DLL

```

The function call_counter_sec is:
1- Input :Sec_count DWORD variable
2- Output: return Sec_count
3- If (Pointer to the shard memory <> NULL)
    Pointer to the shard memory -> RateOfCountsPerSecond32 =
    Sec_count;      Pointer to the shard memory -> Numberofitems32+=
    Sec_count;
    // the number of bytes received per client
4- END

The function call_counter_hour is:
1-Input: Hour_count DWORD variable
2- Output: return Hour_count
3- If (Pointer to the shard memory <> NULL) THEN
    Pointer_stru_insharedmem -> RateOfCountsPerHour +=Hour_count;
4- END

```

Algorithm (4) - Client Performance Counters Functions

```

1- set the timer to time inertial 20 second,1 hour, see windows procedure of
   the client module [1 MSDN]
2- Set Timer (hwnd(1), (2)Timer_DBCounterSec_ID, (3)20000,(4) NULL)
   //each 20 second interval
3- Set Timer (hwnd, Timer_DBCounterHour_ID, 3600000, NULL)
   //360 =1 hour interval
   // (1) handle (pointer to window)
   // (2) timer identifier
   // (3) time-out value
   // (4) timer procedure
4- IF( NOT (WM_TIMER) ) THEN GOTO STEP 6
   a- IF ( Timer_DBCounterSec_ID<> wParam) GOTO STEP 5
   b- memset(Modules_Counters1,0,sizeof(MODULES_COUNTERS));
   c- Modules_Counters1.ModulesTransPerSec=CounterDB/20;
   d- Fields [0] =mcModulesTransPerSec;
      //need to be saved in an array to check it in debug process
   e- call_counter_sec(Modules_Counters1.ModulesTransPerSec);
      //call for client performance DLL function(1) in Algorithm (4)

5- IF( Timer_DBCounterHour_ID <> wParam) THEN GOTO STEP 6
   a- memset(&Modules_Counters1,0,sizeof(MODULES_COUNTERS));
   b- Modules_Counters1.ModulesTransPerHour=CounterHour;
   c- Fields [1] =mcModulesTransPerHour;
      //need to be saved in an array to check it in debug process
   d- call_counter_hour(Modules_Counters1.ModulesTransPerHour);
      //call for client performance DLL function(2) in Algorithm (4)
   e- CounterHour=0;
6- END

```

Any update
for counterDB
is done in
critical section

Algorithm (5) – show how to call Performance counter from Clients Modules program

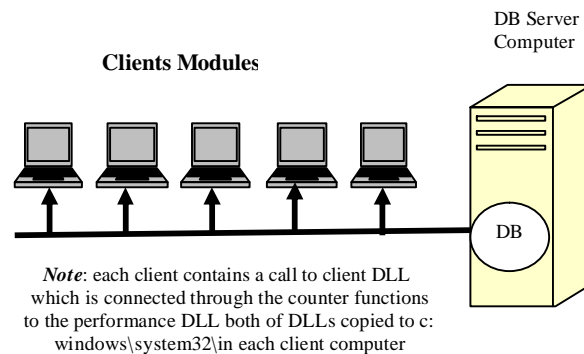


Figure (7)-DB server with Clients Modules

The following flowchart describes the DB Server work:

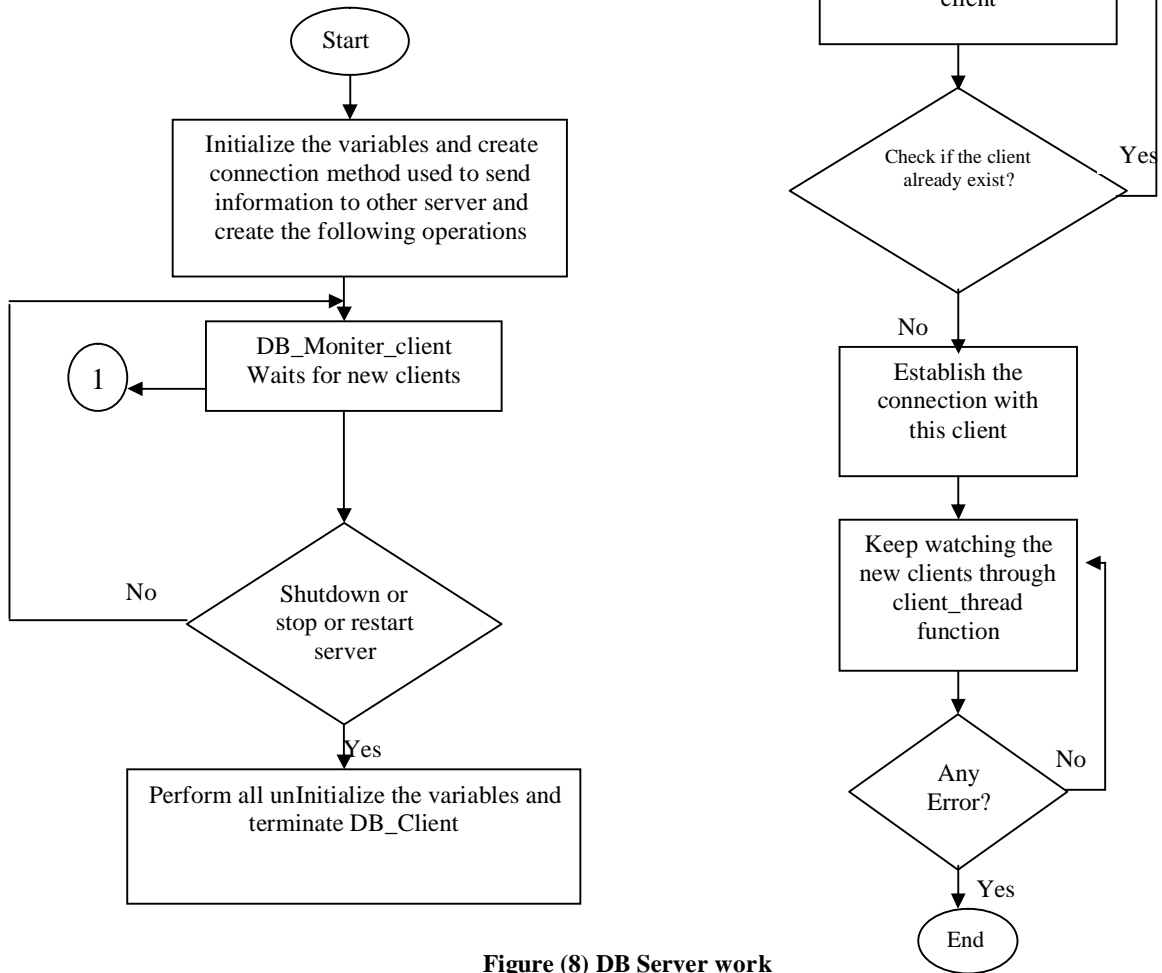


Figure (8) DB Server work

The following flowchart describes the work of the DB Server Client module:

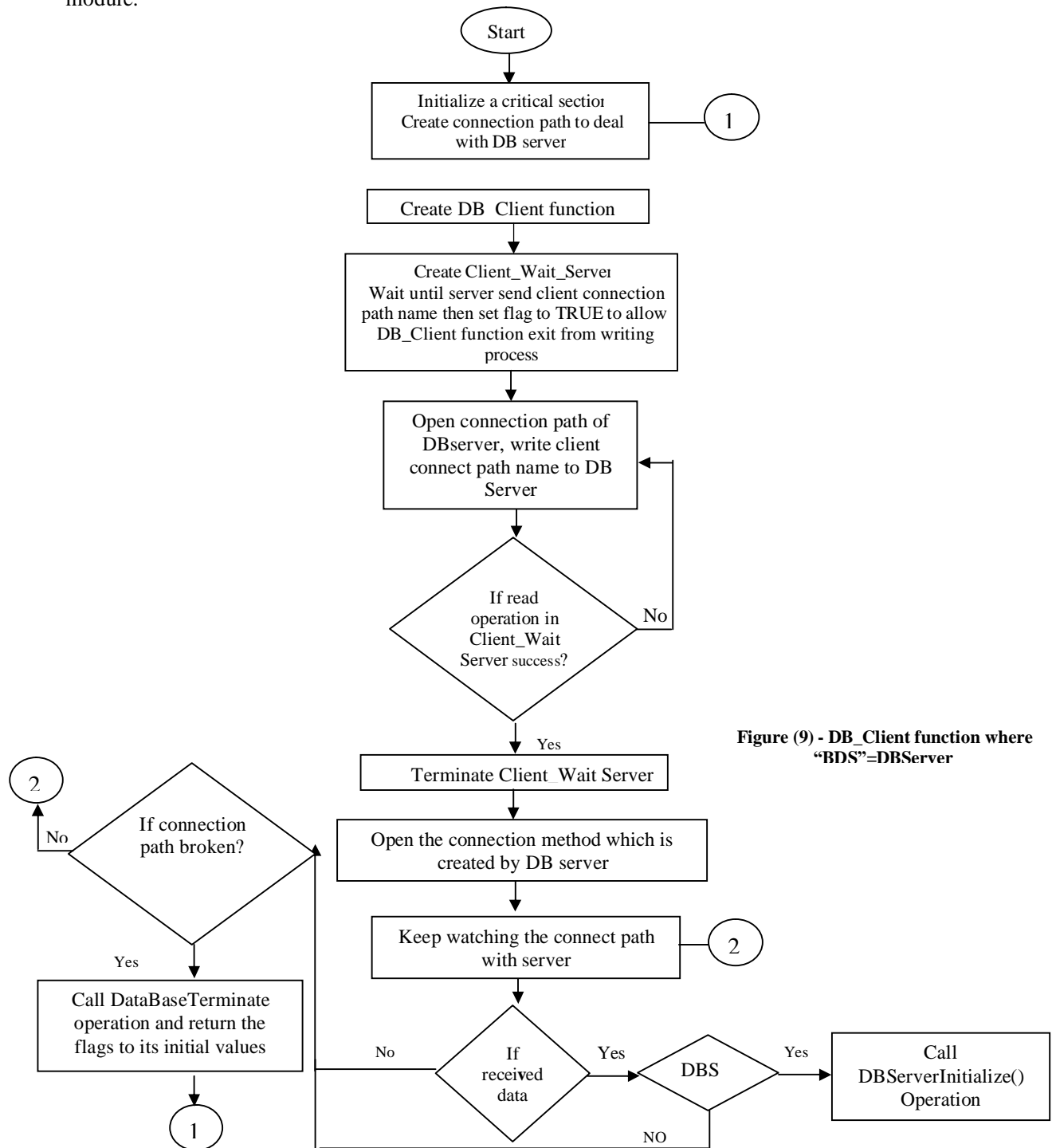


Figure (9) - DB_Client function where
"BDS"=DBServer

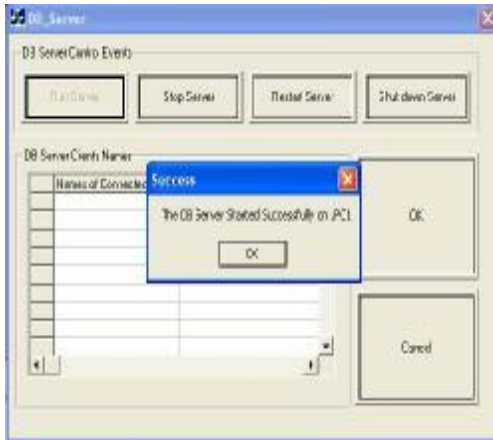


Figure (10) - Database Server Start

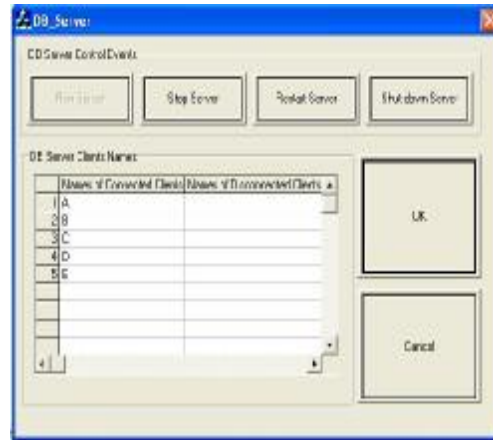


Figure (11) - Database Server and connected client:

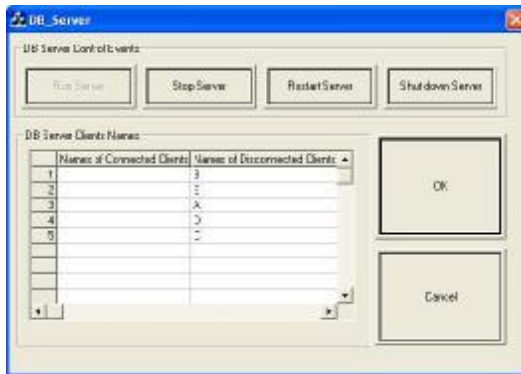


Figure (12) - Database Server and disconnected clients

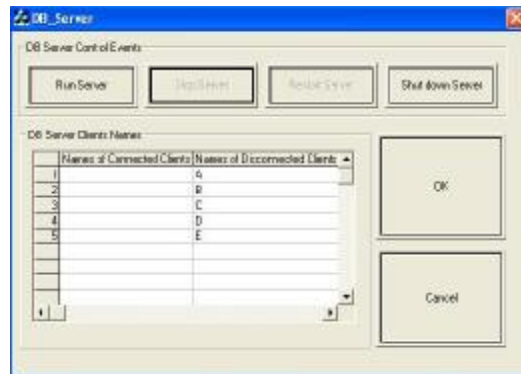


Figure (13) - Database Server Stopped

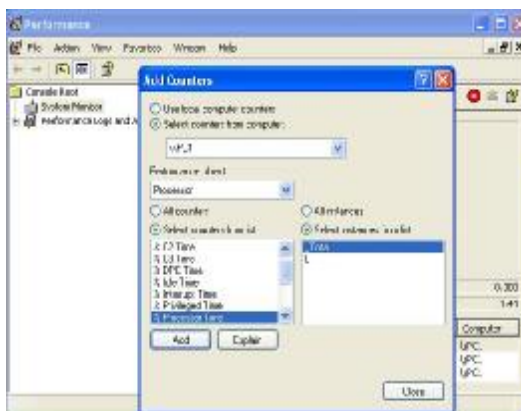


Figure (14) Performance Monitor and add counter

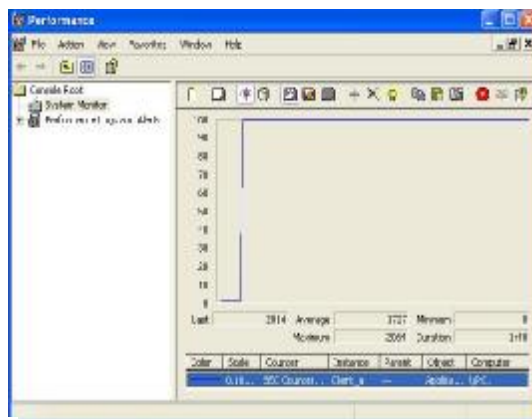


Figure (15) - Client_a performance counter

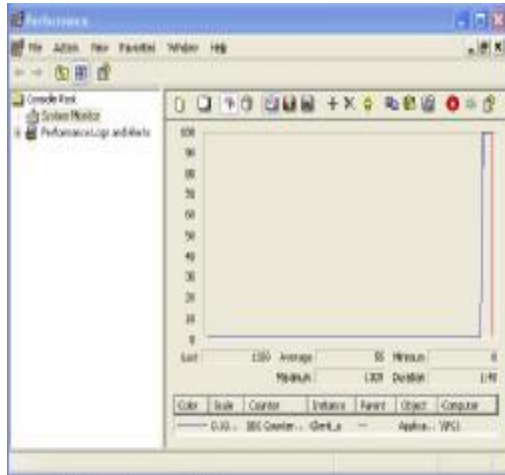


Figure (16)-Client_a after amendment

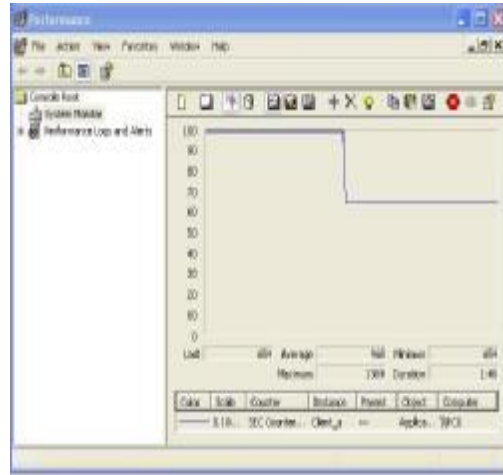


Figure (17)-Client_a after amendment

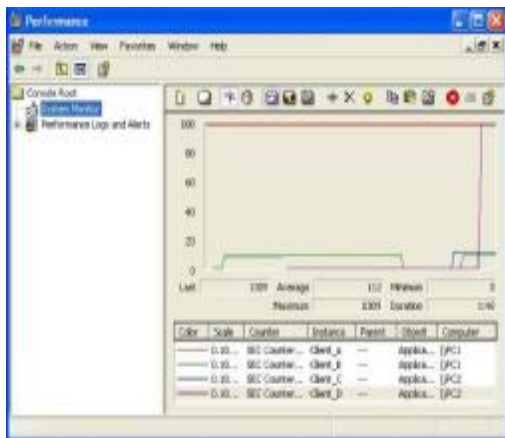


Figure (18)-Clients with Second performance Counter

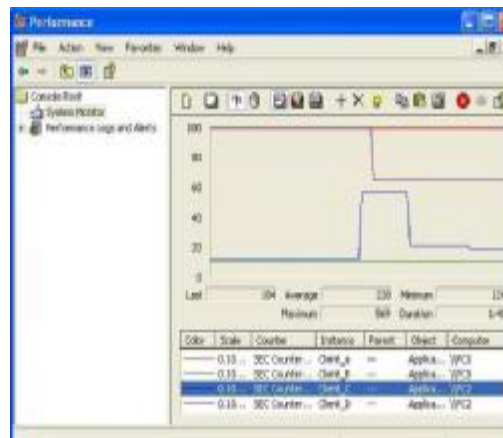


Figure (19)-Client_a appear to the peak of DB

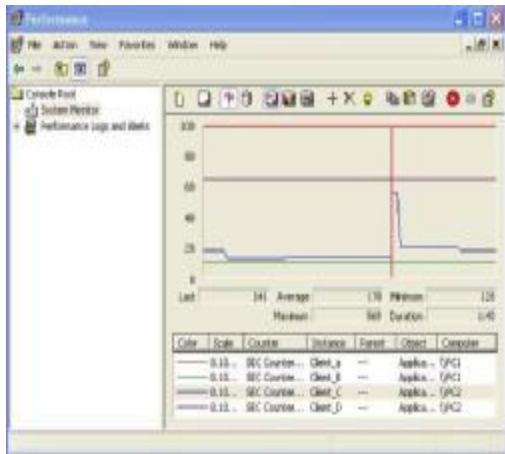


Figure (20)-Client_D began to be more stable

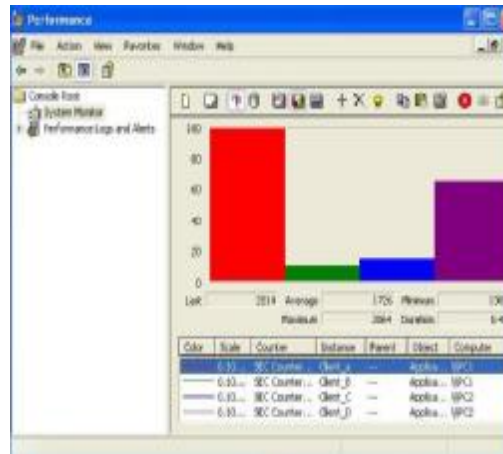


Figure (21)-Histogram for all four Clients

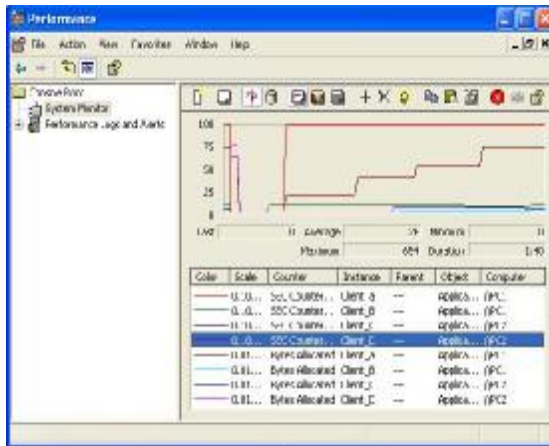


Figure (22)-Second & Bytes counters for all four

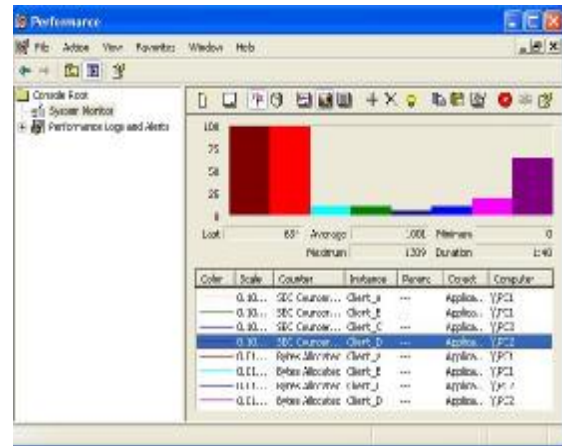


Figure (23)-Histogram for Second and bytes counter

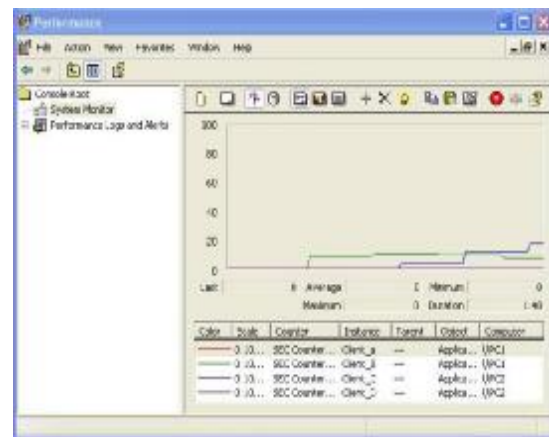


Figure (24)-Clients with Second performance Counter after amendment client_a

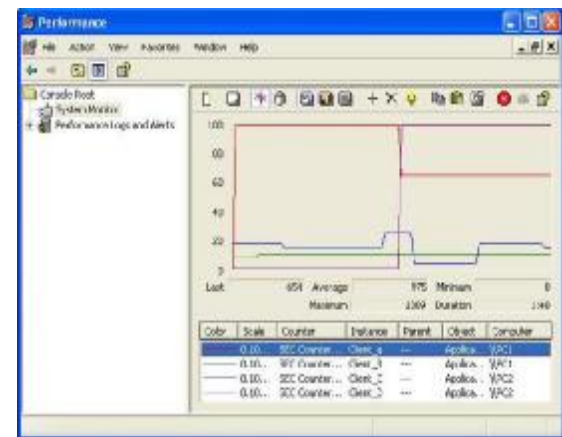


Figure (25)-clients reach the peak after 35~45 second after amendment

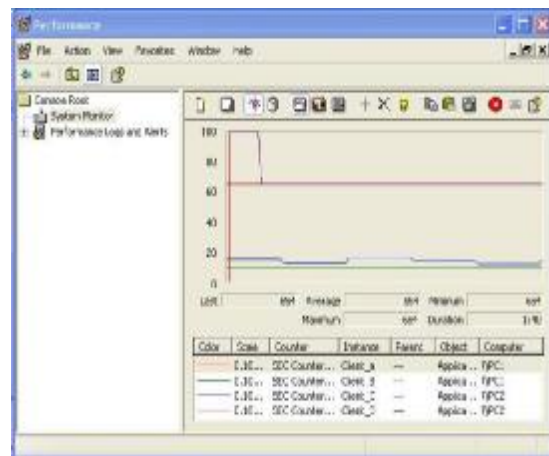


Figure (26)-Client-a after amendment it leaves peak after 35~45 Second

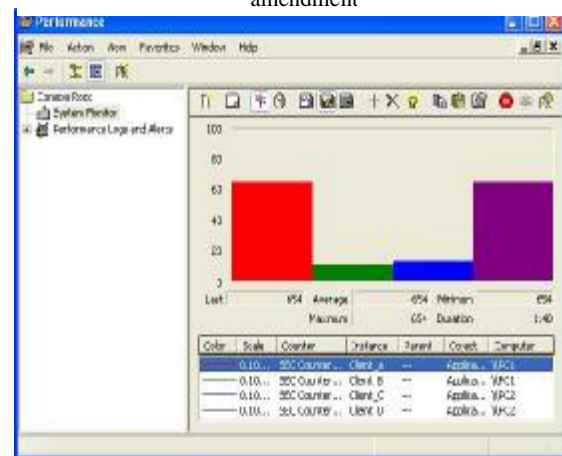


Figure (27)-Histogram for Clients with Second Counter after amendment

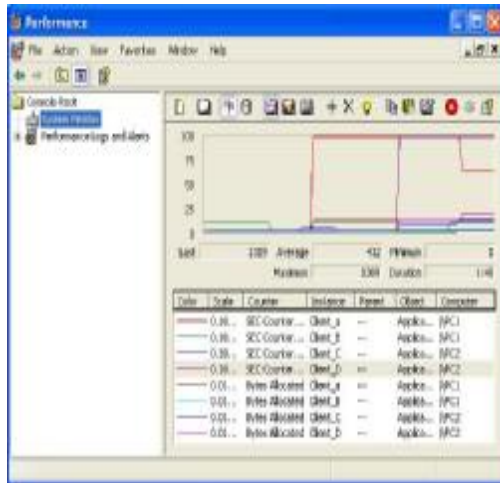


Figure (28)-Second & Bytes counters for all four Clients after amendment

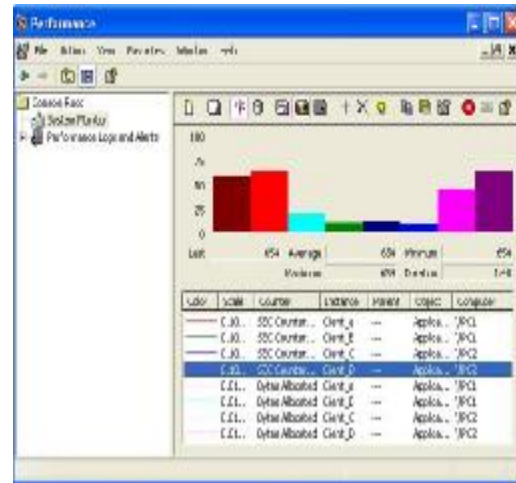


Figure (29)-Histogram for Second and bytes counters after amendment

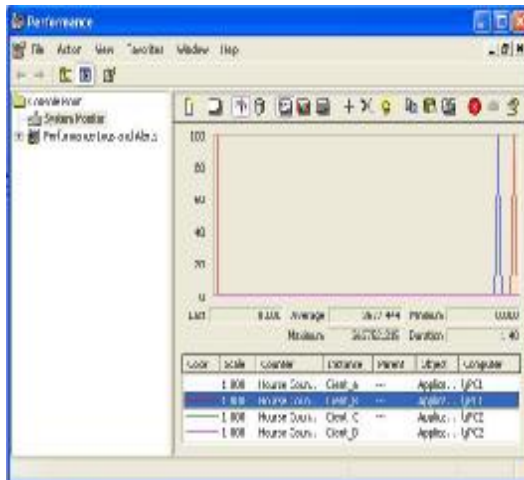


Figure (30)-Hours counters for (a, b) Clients

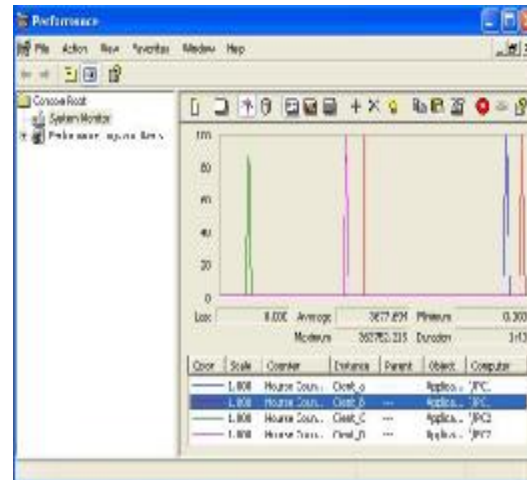


Figure (31)-Hours counters for all four Clients

