Identifying High CPU SQL Processes

By Jeff Stevenson

Contents

Executive Summary 3
Abstract 4
Purpose 4
Problem 5
Solution Description 6
Solution 7
Conclusions 19
About the author 20
Document Information 21
Appendix A. Additional Tips 22

Table of Figures

Figure 1 Adding thread instances 7
Figure 2 Perfmon showing thread instances 8
Figure 3 Adding ID threads 9
Figure 4 Perfmon showing thread-ID thread correlation 9
Figure 5 Perfmon showing high CPU thread selected 10
Figure 6 Perfmon showing thread-ID thread correlation 11
Figure 7 Query Analyzer results correlating KPID-SPID 11
Figure 8 Enterprise Manager showing process' last command 12
Figure 9 Enterprise Manager showing cursor operation 13
Figure 10 Configuring Profiler trace 14
Figure 11 Configuring Profiler events 15
Figure 12 Configuring Profiler Columns 16
Figure 13 Configuring Profiler filter 17
Figure 14 Profile trace results 18

Section 1

Executive Summary

As a SQL Server administrator you are probably familiar with this scenario: The phone rings and it is one of your users saying "the system seems really slow today", or if you have monitoring and alerting set up you get an email telling you the CPU(s) on your SQL server is maxed at 100%. Regardless of the method of alert, you have to figure out what is causing the problem. You investigate and find that the sqlservr.exe process is using most, if not all of the available CPU cycles. Beyond that however, you have no idea of exactly *what* in SQL Server is causing the CPU to be pegged at 100%.

When faced with the issue of identifying the exact runaway process or statement that is bringing your SQL Server to its knees you will find that there is no easy solution. There is no single place in the provided tools to dynamically list the percent of CPU taken by each SQL Server thread process.

Even identifying the specific SQL Server process leaves you grasping for more information. You really want to know the exact code or statement that is soaking up precious CPU cycles. Using the techniques described in this paper you will be able to identify the culprit and react appropriately, adding efficiency and value to your organization.

Section 2

Abstract

Purpose

The purpose of this paper is to describe the methods that can be used to determine what SQL Server process is consuming CPU cycles. Through the use of Performance Monitor, Query Analyzer, Enterprise Manager and Profiler, tools provided with Microsoft Windows and Microsoft SQL Server, the administrator can pinpoint the specific query or operation that is causing high CPU.

Step-by-step instructions, including screen captures, detailing the setup and execution are provided to enable the system administrator to quickly and accurately determine the exact cause and start him on the correct course to fix the problem. A special section addressing cursors and stored procedures is included to deal with situations where the statement causing the problem is not readily apparent as is common in applications such as JD Edwards EnterpriseOne.

A bonus section is included with additional tips to make the methods described in the paper quick and easy to repeat. Several additional tips related to Oracle JD Edwards EnterpriseOne are included.

Total Pages - 22

Section 3

Problem

When a high CPU condition caused by SQL Server presents itself on your server, there is no quick, simple way to determine exactly what SQL Server operation is causing the problem.

The steps to get to the answer are not readily apparent, not integrated and do not lend themselves to an easy approach to a solution. We simply need a way to pinpoint the exact operation within SQL Server that is causing the high CPU condition.

Section 4

Solution Description

To achieve our goal of determining the statement or operation causing high SQL Server CPU we will make use of the multiple SQL Server tools necessary to get to the detail level we desire.

We will follow a troubleshooting trail from Performance Monitor to Query Analyzer to Enterprise Manager to Profiler to get to the bottom of the problem. When completed, you will be able to quickly determine which operation is causing the high CPU problem.

Each step in the solution has detailed setup and execution instructions, complete with screen shots and descriptions.

In order to determine exactly what statement or operation is causing high CPU on the SQL server, it is necessary to follow a path that looks like this:

Instance/Thread->ID Thread->KPID->SPID

To get from the application that can tell you there is a high CPU issue (Performance Monitor) to where you can view the command causing the issue (Enterprise Manager or Profiler) you must follow the trail mentioned above.

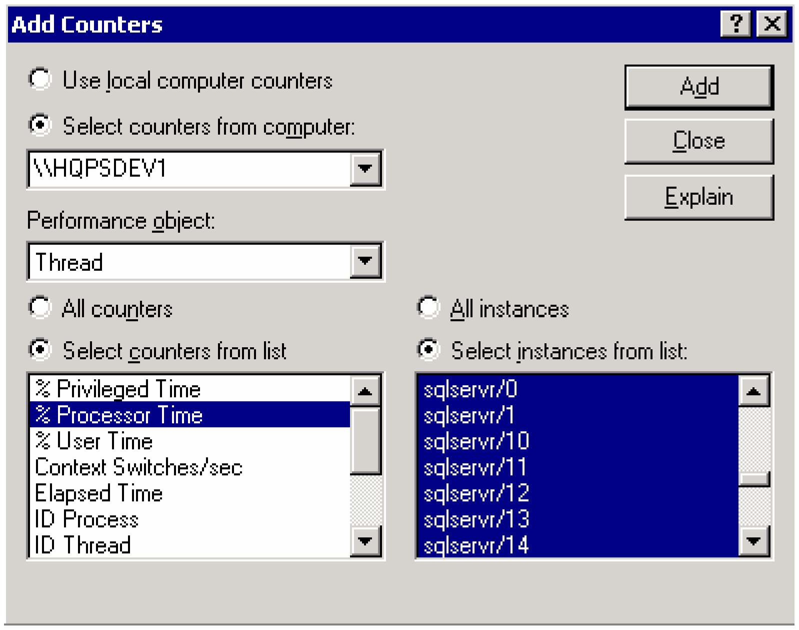
To view the statement or operation causing the problem, it is necessary to know the SPID (Server Process ID). To get the SPID you must know the KPID (Kernel Process ID). To get the KPID you must have the ID Thread. To get the ID Thread, you must know the Thread or Instance number. So, you follow the trail above to achieve the goal of determining the exact command causing your high CPU state.

Section 5

Solution

Setup

In Performance Monitor, you will view the SQL threads' CPU consumption. Open perfmon and add the appropriate counters by selecting the "thread" performance object, the counter object “% Processor Time”, and all of the "sqlservr" instances.



Use local computer counters [✓ Select counters from computer:]

\\HQPSDEV1

Performance object:

Thread

All counters [✓ Select counters from list]

All instances [✓ Select instances from list:]

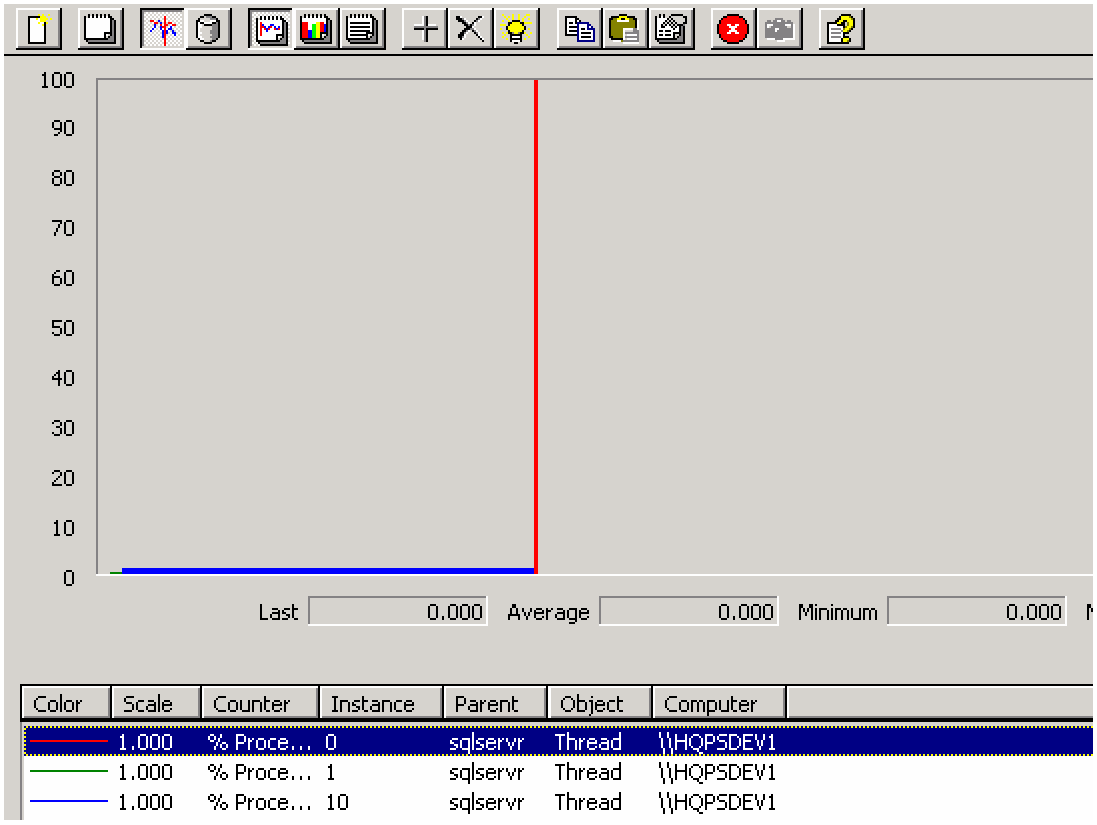
% Privileged Time
% Processor Time
% User Time
Context Switches/sec
Elapsed Time
ID Process
ID Thread

sqlservr/0
sqlservr/1
sqlservr/10
sqlservr/11
sqlservr/12
sqlservr/13
sqlservr/14

Figure 1 Adding thread instances

This will display %Processor Time by thread instance number and allow you to identify the high CPU thread.

The Performance Monitor window will show each instance as a line in graph mode or as bars in histogram mode. The instance counters will be listed at the bottom as well.

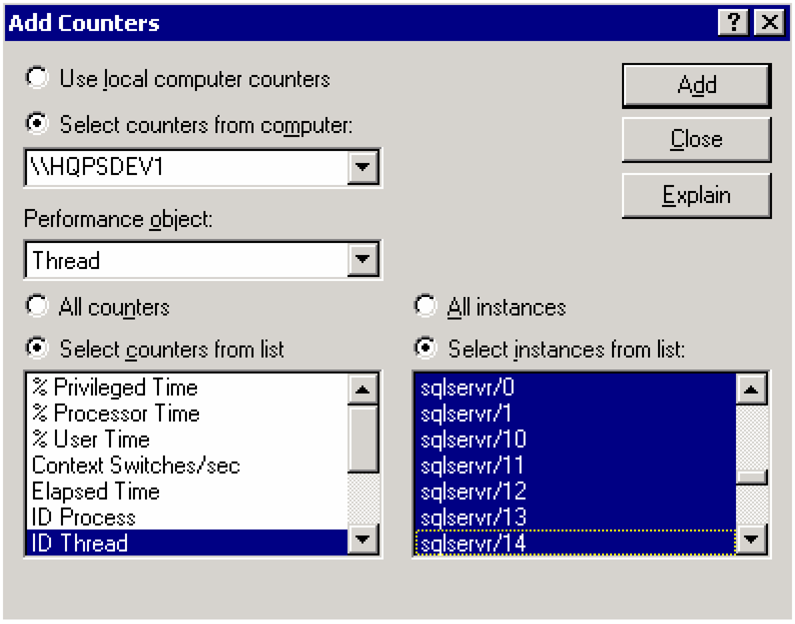


Last 0.000 Average 0.000 Minimum 0.000

Color	Scale	Counter	Instance	Parent	Object	Computer
🟥	1.000	% Proce...	0	sqlservr	Thread	\\HQPSDEV1
🟩	1.000	% Proce...	1	sqlservr	Thread	\\HQPSDEV1
🟦	1.000	% Proce...	10	sqlservr	Thread	\\HQPSDEV1

Figure 2 Perfmon showing thread instances

Start a second session of perfmon, and click the View/Report button. Then add the performance object "thread", the counter object "ID Thread", and all the "sqlservr" instances.



Use local computer counters [✓ Select counters from computer:] \\HQPSDEV1

Performance object: Thread

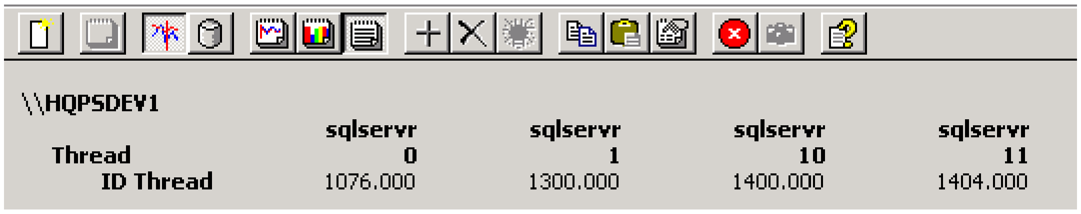
All counters [✓ Select counters from list]

All instances [✓ Select instances from list:]

% Privileged Time	sqlservr/0
% Processor Time	sqlservr/1
% User Time	sqlservr/10
Context Switches/sec	sqlservr/11
Elapsed Time	sqlservr/12
ID Process	sqlservr/13
ID Thread	sqlservr/14

Figure 3 Adding ID threads

This will appear as a tabular report in Performance Monitor with the ID thread number listed below the corresponding thread instance number.



```
\\HQPSDEV1
                        sqlservr     sqlservr     sqlservr     sqlservr
   Thread                      0            1           10           11
          ID Thread     1076.000     1300.000     1400.000     1404.000
```

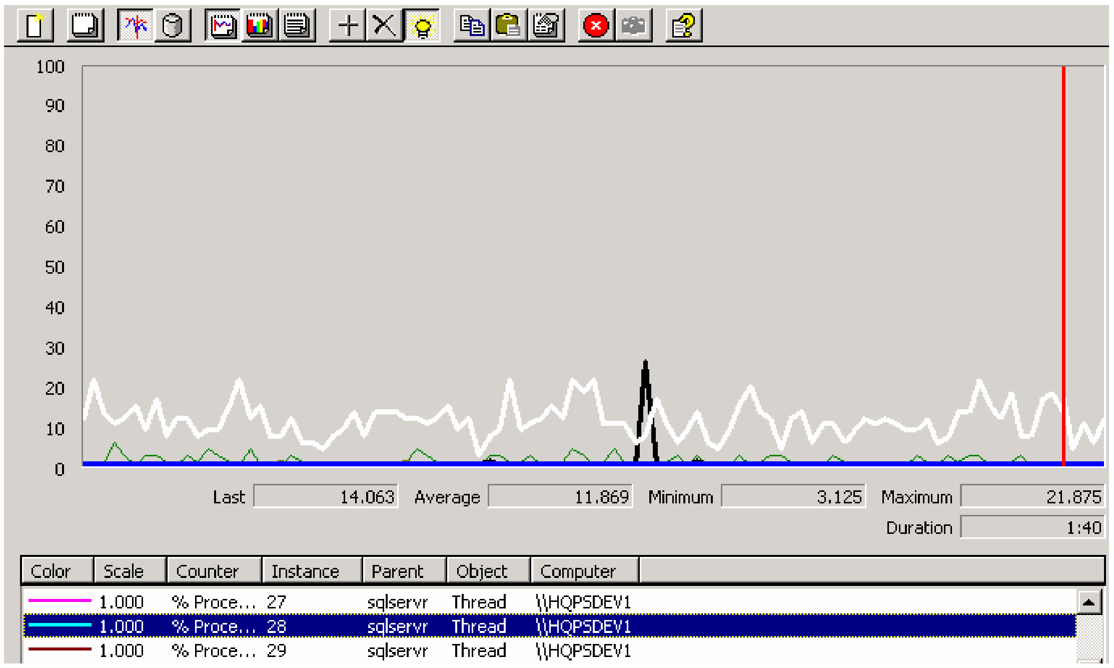
Figure 4 Perfmon showing thread-ID to thread correlation

Execution

When the occasion arises where you must find out what SQL Server SPID is causing the high CPU, execute the steps below.

In Performance Monitor session #1 find the thread instance that shows **sustained** high CPU %. Note that on a busy system there may be several thread instances showing high CPU %. If you are troubleshooting a long-running process, the obvious thread will show after a prolonged period of watching. You may need to switch to histogram view in Performance Monitor if there are a large number of active SQL Server threads. Note that it may not be necessary to choose one thread as the process can be spread among multiple threads that all correlate back to a single SPID. For example you may find that instance 28 and instance 50, while having different ID Threads, may have the same SPID and the SPID is what we are trying to find.

Once you choose a thread instance, double-click on the graph line (or bar in histogram view), highlighting the instance in the bottom panel and enabling you to identify the insance number. Note the instance number, in this case instance 28.

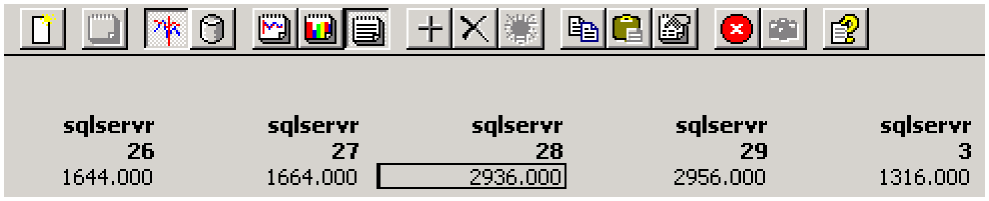


Last 14.063 Average 11.869 Minimum 3.125 Maximum 21.875
Duration 1:40

Color	Scale	Counter	Instance	Parent	Object	Computer
🟪	1.000	% Proce...	27	sqlservr	Thread	\\HQPSDEV1
🟦	1.000	% Proce...	28	sqlservr	Thread	\\HQPSDEV1
🟥	1.000	% Proce...	29	sqlservr	Thread	\\HQPSDEV1

Figure 5 Perfmon showing high CPU thread selected

Switch to performance monitor #2 and find the instance number noted above in the Thread row. Note the ID Thread listed below the thread. In this case the thread is 28 and the ID Thread is 2936.



sqlservr 26	sqlservr 27	sqlservr 28	sqlservr 29	sqlservr 3
1644.000	1664.000	2936.000	2956.000	1316.000

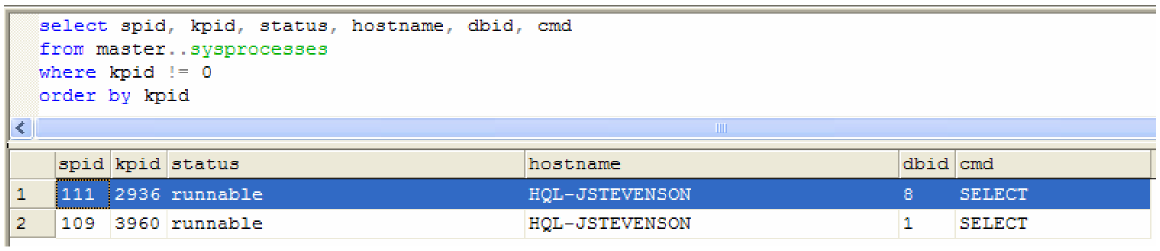
Figure 6 Perfmon showing thread-ID to thread correlation

The ID Thread correlates with KPID (Kernel Process ID). Once we have the KPID we are one step closer to our goal of getting the SPID of the high CPU SQL process. The KPID to SPID relationship can be determined by executing the following SQL statement on the affected server while the offending process is running:

select spid, kpid, status, hostname, dbid, cmd from master..sysprocesses

where kpid != 0 order by kpid

The results show which SPID (Server Process ID) correlates with the KPID. In our case KPID 2936 is associated with SPID 111.



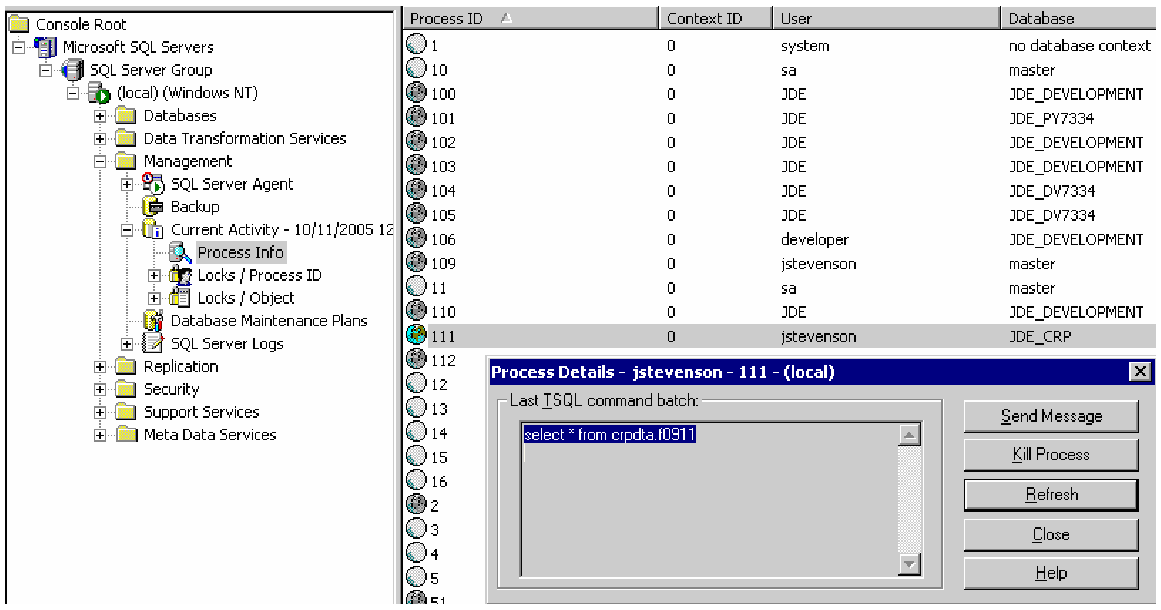
```
select spid, kpid, status, hostname, dbid, cmd
from master..sysprocesses
where kpid != 0
order by kpid
```

	spid	kpid	status	hostname	dbid	cmd
1	111	2936	runnable	HQL-JSTEVENSON	8	SELECT
2	109	3960	runnable	HQL-JSTEVENSON	1	SELECT

Figure 7 Query Analyzer results correlating KPID-SPID

With this information you can now use Enterprise Manger, sp_who2 or other tools to find out more about what the SPID is doing.

In Enterprise Manager select the appropriate SQL Server then expand Management/Current Activity/Process Info. In the list of Process IDs find the SPID you determined in the previous step. You can double-click the SPID record to see exactly what the last batch command was. In the case of SPID 111, we see that a select statement was being run against CRPDTA.F0911, a table with over 8 million records, causing significant CPU usage by SQL Server.



- Console Root
  - Microsoft SQL Servers
    - SQL Server Group
      - (local) (Windows NT)
        - Databases
        - Data Transformation Services
        - Management
          - SQL Server Agent
          - Backup
          - Current Activity - 10/11/2005 12
            - Process Info
            - Locks / Process ID
            - Locks / Object
          - Database Maintenance Plans
          - SQL Server Logs
        - Replication
        - Security
        - Support Services
        - Meta Data Services

Process ID	Context ID	User	Database
1	0	system	no database context
10	0	sa	master
100	0	JDE	JDE_DEVELOPMENT
101	0	JDE	JDE_PY7334
102	0	JDE	JDE_DEVELOPMENT
103	0	JDE	JDE_DEVELOPMENT
104	0	JDE	JDE_DV7334
105	0	JDE	JDE_DV7334
106	0	developer	JDE_DEVELOPMENT
109	0	jstevenson	master
11	0	sa	master
110	0	JDE	JDE_DEVELOPMENT
111	0	jstevenson	JDE_CRP
112			
12			
13			
14			
15			
16			
2			
3			
4			
5			
51			

Process Details - jstevenson - 111 - (local)

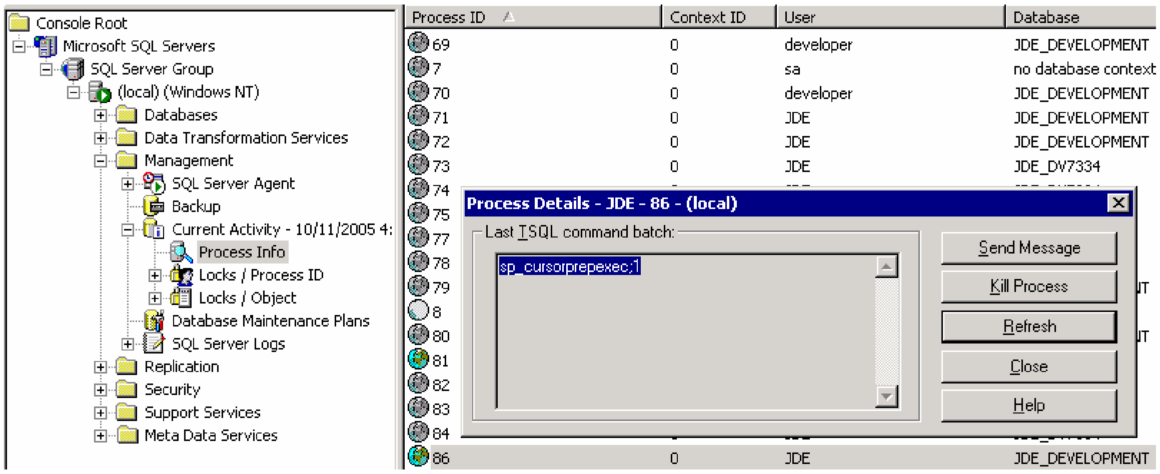
Last TSQL command batch:

```
select * from crpdta.f0911
```

Figure 8 Enterprise Manager showing process' last command

Cursor Issues

An issue arises however, if the activity you are trying to identify is typical of EnterpriseOne, which makes extensive use of cursors. When correlating back to a SPID, the activity you will likely see is a stored procedure cursor command such as sp_cursorprepexec or sp_cursorfetch instead of an actual SQL statement. All this indicates is that a cursor is being executed and tells you little about the underlying activity that is causing your CPU increase.



Process ID	Context ID	User	Database
69	0	developer	JDE_DEVELOPMENT
7	0	sa	no database context
70	0	developer	JDE_DEVELOPMENT
71	0	JDE	JDE_DEVELOPMENT
72	0	JDE	JDE_DEVELOPMENT
73	0	JDE	JDE_DV7334
74			
75			
77			
78			
79			
8			
80			
81			
82			
83			
84			
86	0	JDE	JDE_DEVELOPMENT

Process Details - JDE - 86 - (local)

Last TSQL command batch:

```
sp_cursorprepexec;1
```

Figure 9 Enterprise Manager showing cursor operation

You can still get to the root of your problem if the SPID correlates back to a cursor; it will just take a little more work.

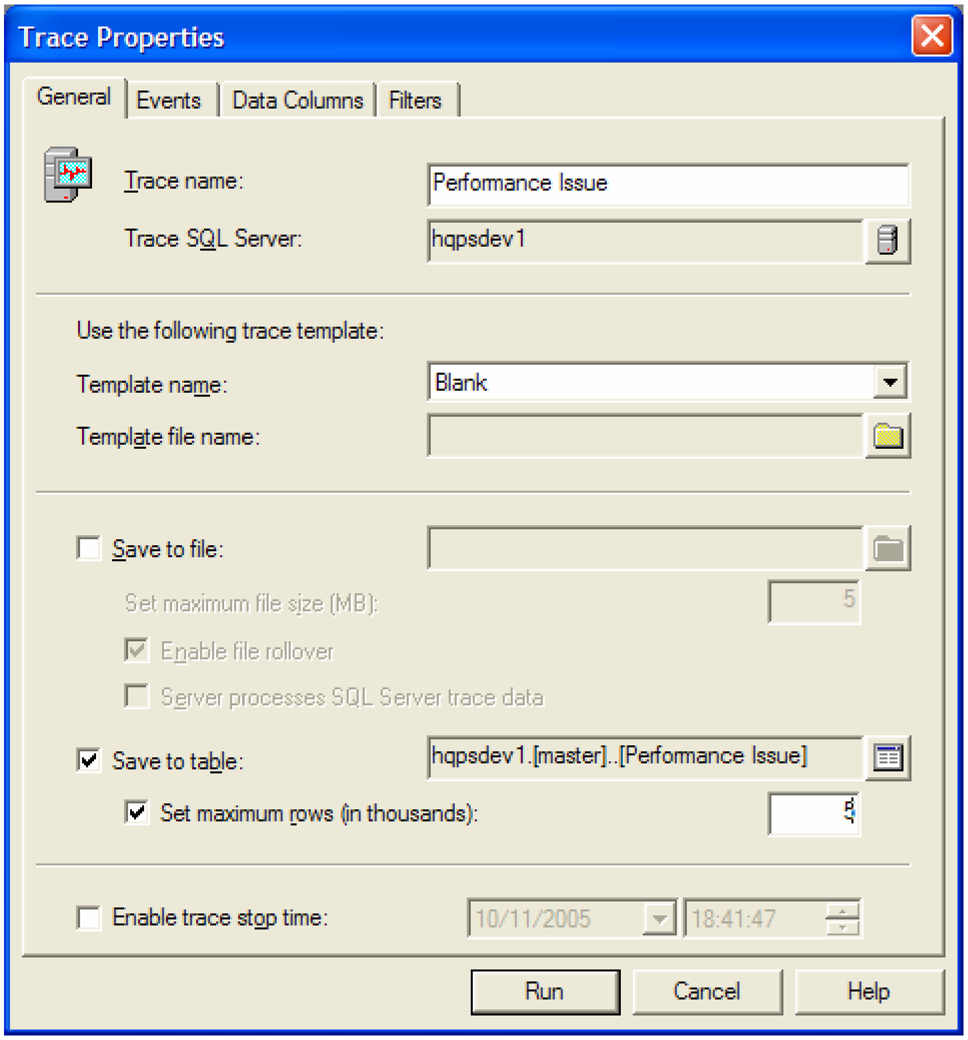
Profiler Setup

To find what activity is associated with the cursor operation you will use SQL Server Profiler, a performance analysis tool that comes with SQL Server and can be used to trace events and activities on your SQL server.

Since Profiler tracing adds overhead, we must very narrowly define our criteria for analysis in Profiler in order to not overburden the server being monitored. Since we are still operating from the assumption that we are searching for the root cause of an activity causing high CPU on SQL Server, it would not be helpful to add to the load unnecessarily.

Open Profiler and select New/Trace from the File menu. Select the affected server from the dropdown list.

On the "General" tab enter the information about your trace. Name your trace, select Template Name "Blank", select a location to save the trace and set the maximum rows. Since we have the option to save the trace information to a table we shall do so.



Trace name: Performance Issue

Trace SQL Server: hqpsdev1

Use the following trace template:

Template name: Blank

Template file name:

☐ Save to file:

Set maximum file size (MB): 5

☑ Enable file rollover

☐ Server processes SQL Server trace data

☑ Save to table: hqpsdev1.[master]..[Performance Issue]

☑ Set maximum rows (in thousands): 5

☐ Enable trace stop time: 10/11/2005 18:41:47

Figure 10 Configuring Profiler trace

Select the "Events" tab and add the following events:

Cursors: CursorClose, CursorExecute, CursorOpen,

Performance: Show Plan Text

Stored Procedures: RPC:Completed, RPC:Starting, SP: StmtCompleted, SP: StmtStarting

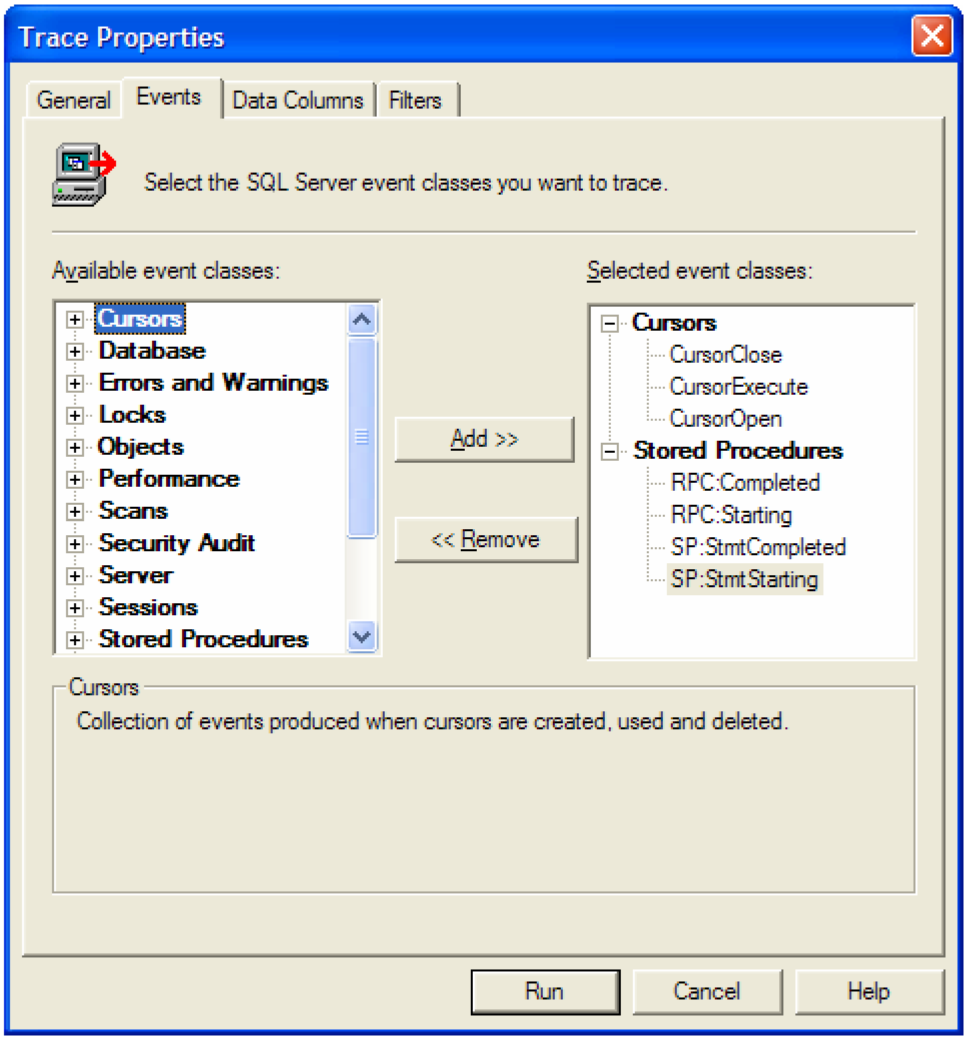
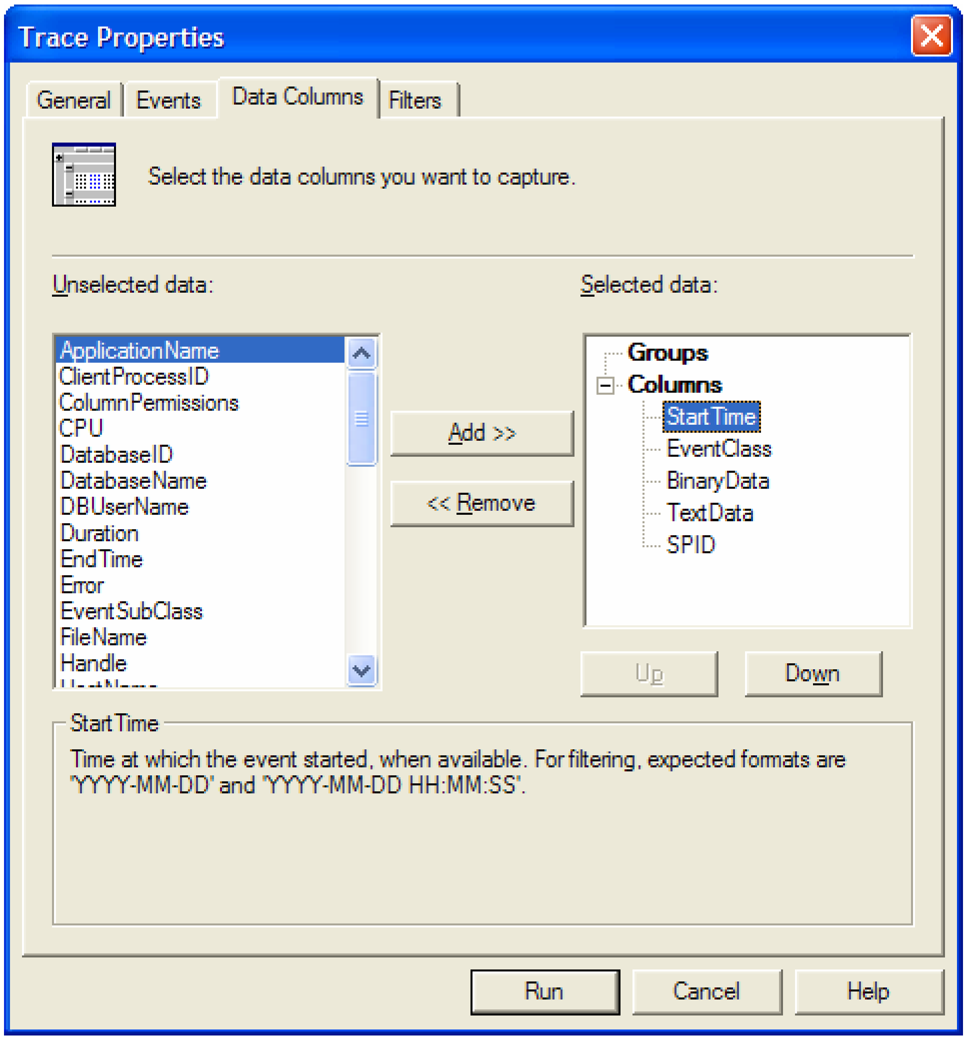


Figure 11 Configuring Profiler events

These are the events that we will trace in Profiler.

Select the "Data Columns" tab and add the following to the "Columns" section:

Start Time, BinaryData, TextData. These values will join EventClass and SPID which are already present.



Select the data columns you want to capture.

Unselected data:

- ApplicationName
- ClientProcessID
- ColumnPermissions
- CPU
- DatabaseID
- DatabaseName
- DBUserName
- Duration
- EndTime
- Error
- EventSubClass
- FileName
- Handle

Selected data:

- Groups
- Columns
  - StartTime
  - EventClass
  - BinaryData
  - TextData
  - SPID

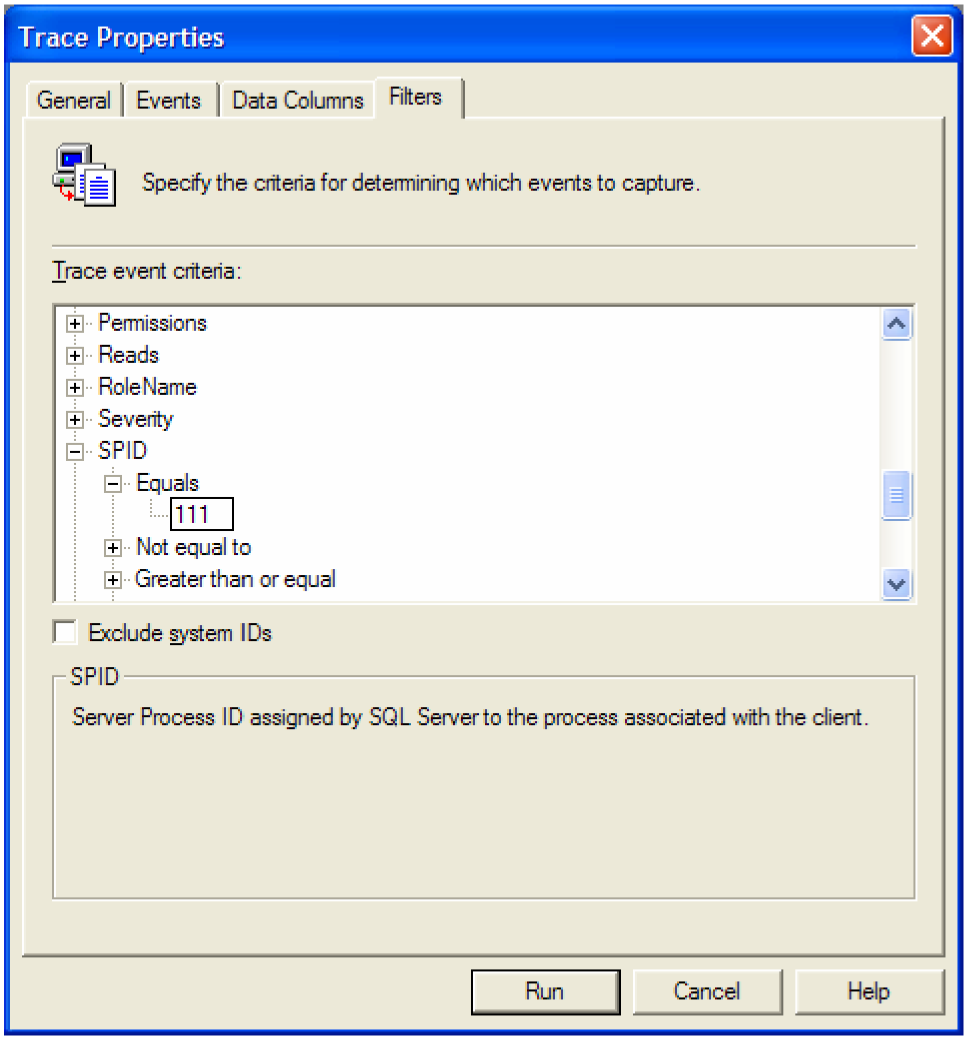
StartTime

Time at which the event started, when available. For filtering, expected formats are 'YYYY-MM-DD' and 'YYYY-MM-DD HH:MM:SS'.

Figure 12 Configuring Profiler Columns

These are the data items that will be visible in the trace results.

Select the "Filters" tab to set up Profiler to only show the events from the SPID that we discovered in the earlier sections to be the offending process. In the example picture we will presume that the offending SPID was found to be 111. Enter 111 in the SPID/Equals filter to see only events associated with SPID 111.



Specify the criteria for determining which events to capture.

Trace event criteria:

- Permissions
- Reads
- RoleName
- Severity
- SPID
  - Equals
    - 111
  - Not equal to
  - Greater than or equal

☐ Exclude system IDs

SPID

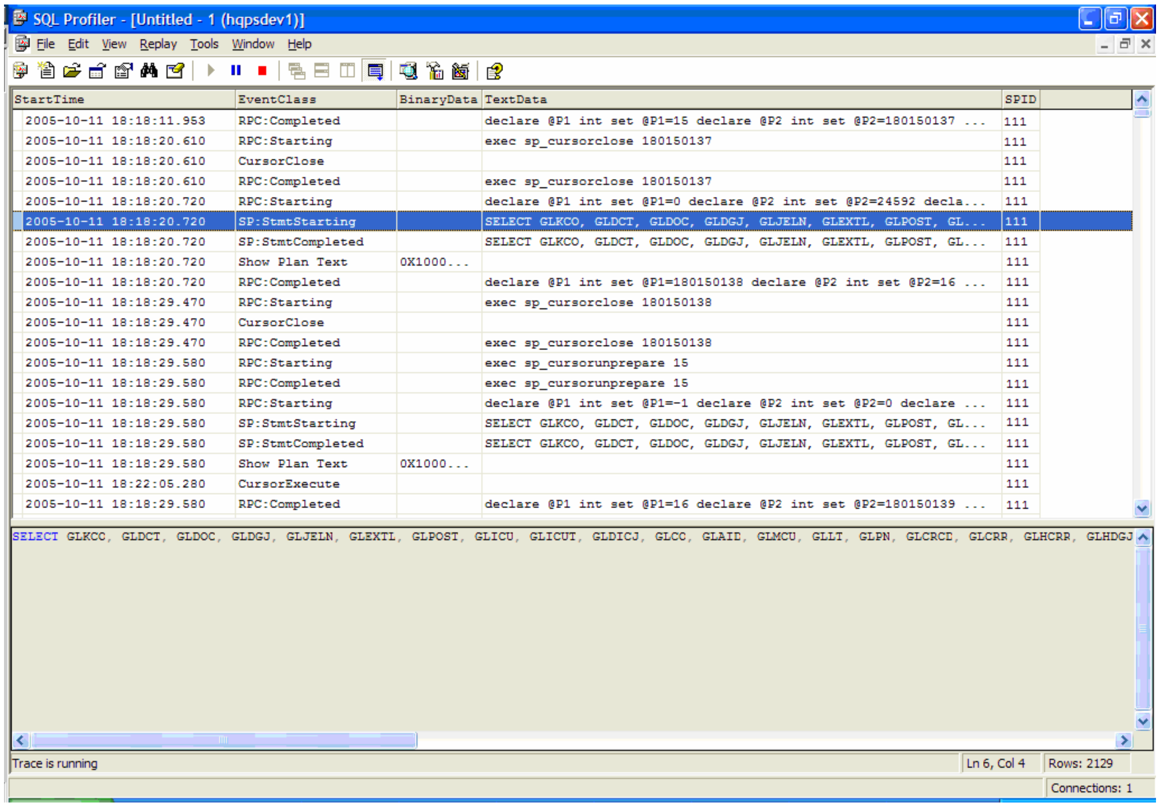
Server Process ID assigned by SQL Server to the process associated with the client.

Figure 13 Configuring Profiler filter

Profiler Execution

Click the Run button and you should see data scrolling by. If the operation you are trying to trace has already gone into the use of cursors you may have difficulties telling exactly what is causing the high CPU. If you are fortunate, a quick examination of the trace results will point to the exact cause.

The trace below shows several select statements interspersed with the cursor executions.


SQL Profiler - [Untitled - 1 (hqpsdev1)]
File Edit View Replay Tools Window Help

StartTime	EventClass	BinaryData	TextData	SPID
2005-10-11 18:18:11.953	RPC:Completed		declare @P1 int set @P1=15 declare @P2 int set @P2=180150137 ...	111
2005-10-11 18:18:20.610	RPC:Starting		exec sp_cursorclose 180150137	111
2005-10-11 18:18:20.610	CursorClose			111
2005-10-11 18:18:20.610	RPC:Completed		exec sp_cursorclose 180150137	111
2005-10-11 18:18:20.720	RPC:Starting		declare @P1 int set @P1=0 declare @P2 int set @P2=24592 decla...	111
2005-10-11 18:18:20.720	SP:StmtStarting		SELECT GLKCO, GLDCT, GLDOC, GLDGJ, GLJELN, GLEXTL, GLPOST, GL...	111
2005-10-11 18:18:20.720	SP:StmtCompleted		SELECT GLKCO, GLDCT, GLDOC, GLDGJ, GLJELN, GLEXTL, GLPOST, GL...	111
2005-10-11 18:18:20.720	Show Plan Text	0X1000...		111
2005-10-11 18:18:20.720	RPC:Completed		declare @P1 int set @P1=180150138 declare @P2 int set @P2=16 ...	111
2005-10-11 18:18:29.470	RPC:Starting		exec sp_cursorclose 180150138	111
2005-10-11 18:18:29.470	CursorClose			111
2005-10-11 18:18:29.470	RPC:Completed		exec sp_cursorclose 180150138	111
2005-10-11 18:18:29.580	RPC:Starting		exec sp_cursorunprepare 15	111
2005-10-11 18:18:29.580	RPC:Completed		exec sp_cursorunprepare 15	111
2005-10-11 18:18:29.580	RPC:Starting		declare @P1 int set @P1=-1 declare @P2 int set @P2=0 declare ...	111
2005-10-11 18:18:29.580	SP:StmtStarting		SELECT GLKCO, GLDCT, GLDOC, GLDGJ, GLJELN, GLEXTL, GLPOST, GL...	111
2005-10-11 18:18:29.580	SP:StmtCompleted		SELECT GLKCO, GLDCT, GLDOC, GLDGJ, GLJELN, GLEXTL, GLPOST, GL...	111
2005-10-11 18:18:29.580	Show Plan Text	0X1000...		111
2005-10-11 18:22:05.280	CursorExecute			111
2005-10-11 18:18:29.580	RPC:Completed		declare @P1 int set @P1=16 declare @P2 int set @P2=180150139 ...	111

```
SELECT GLKCO, GLDCT, GLDOC, GLDGJ, GLJELN, GLEXTL, GLPOST, GLICU, GLICUT, GLDICJ, GLCC, GLAID, GLMCU, GLLT, GLPN, GLCRCD, GLCRR, GLHCRR, GLHDGJ
```

```
Trace is running                                        Ln 6, Col 4    Rows: 2129
                                                                       Connections: 1
```

Figure 14 Profile trace results

You should be able to analyze the trace results to determine your high CPU culprit, even if it involves a cursor action. Some detective work may be required but it is still much simpler than guessing what SQL Server process is causing your problem.

Section 6

Conclusions

Using the methods provided in the solution detailed, an administrator will be able to pinpoint the cause of high CPU in SQL Server.

This solution will be quick, efficient, repeatable and will work regardless of whether the operation causing the increased CPU usage is a simple SQL statement or a stored procedure.

The SQL Server administrator now possesses a reliable method to help answer the question: “Why does the system seem slow today”

Section 7

About the author

Jeff Stevenson is a senior technologist with broad experience on SQL Server, WebSphere, Windows, Linux, Apache, Drupal and other technologies but primarily focusing on Oracle's JD Edwards EnterpriseOne product line.

Jeff has over six years of award-winning EnterpriseOne experience and has spent fifteen years in the high-tech world, including a stint in the US Marine Corps working on computer and electronics systems of advanced fighter jets.

A leading member of the EnterpriseOne community, he has given numerous presentations, conducted workshops and technology survival camps both as an employee of JD Edwards/PeopleSoft and independently. He is an active participant on the jdelist.com website, dispensing advice under the pseudonym Brother of Karamazov.

Jeff lives near Washington, DC in Gaithersburg, Maryland with his wife and two girls and likes to participate in short triathlons when not hovering over the keyboard.

Section 8

Document Information

Title	SQL, Identify high CPU SQL Processes
Author	Jeff Stevenson
File Name	SQL, Identify high CPU SQL Processes.doc
Revision	1.00
Technology 1	SQL
Technology 1 Version	SQL 2000/7.0
Technology 2	EnterpriseOne
Technology 2 Version	All
Technology 3	
Technology 3 Version	
Price	

Appendix A. Additional Tips

- To allow fast reaction time to a high CPU condition you can save the Performance Monitor console settings once you have set them as described in the document. By saving each perfmon session you can quickly open the .msc file and begin observing the SQL Server activity.

- By also saving the SQL statement that correlates the KPID to SPID you can also quickly execute it.

- You can save the Profiler template to a file that preserves the settings for quick access. Opening the Profiler template and changing the SPID ID filter to the appropriate SPID is all you have to do to start capturing transaction information.

- The KPID can be associated back to a EnterpriseOne batch job or OneWorld kernel process. Simply take the KPID found in the ID Thread row of perfmon #2 and check task manager on the enterprise server the UBE or BSFN is running on. Sort the list by PID and look for the ID Thread (KPID) number. You can also find the server log for this process in C:\JDEdwardsOneWorld\ddp\B7334\Log\jde_nnnn.log where nnnn is the KPID number.

- In the above, if the KPID is a OneWorld batch job, you can go to Work With Submitted Jobs and enter the KPID in the Process ID field to determine which UBE is causing the issue.

- If the above is a JAS-related process one can try to find the KPID in the Host-PID field of the User List of SAW WEB