PROCESS MONITORING FOR LCD AOI

Wei-Shui Chen(*陳威旭*), *Chiou-Shann Fuh* (*傅楸善*)

Graduate Institute of Networking and Multimedia College of Electrical Engineering and Computer Science, National Taiwan University
E-mail:r96944024@ntu.edu.tw
Dept. of Computer Science and Information Engineering, National Taiwan University
E-mail:fuh@csie.ntu.edu.tw

ABSTRACT

In this paper, we develop a process monitoring algorithm to detect whether the processes running on the LCD defect inspection system fail or are unusually busy and thereafter correct them. We aim to implement this method on the LCD (Liquid Crystal Display) defect inspection system with line-scan cameras. Because current LCD defect inspection systems comprise many CCD (Charge-Coupled Device) cameras and transferring image data to industrial computers with Camera Link for further processing, the algorithm must be simple, fast, and with low computational complexity to fit in with real-time environment. Another important issue in process status determination is how to minimize false alarm rate of the criteria we design.

1. INTRODUCTION

The first step of automatic optical inspection for LCD is image acquisition. As LCDs are mass-produced and their sizes increase quickly, the huge raw image data caught by line-scan cameras need to be sent to each industrial computer with Camera Link for the parallel processing, and each industrial computer runs some specific processes such as image preprocessing. If one of these processes shuts down, it may cause asynchronous data transferring/receiving and interrupt further processing.

In this paper, we develop an automatic process monitoring system for LCD defect inspection installed in each industrial computer based on the observation for CPU (Central Processing Unit) usage and occupation of memory space of the process. We also design the objective criteria to judge whether the process is abnormal.

1.1. Process Monitoring

A process is defined as self-contained package of data and executable procedures which operate on those data, comparable to a task in other known systems. A process can be thought of as comparable to a subroutine in terms of size, complexity, and the way it is used. The difference between processes and subroutines is that processes can be created and destroyed dynamically and can execute concurrently with their creator and other subroutines.

In known data processing systems a Process Control Block (PCB) is used to describe various attributes and status of processes, including the status of resources used by processes. Examples of such resources are files, data storage device, I/O (Input/Output) devices, ports, and so on. In this thesis we utilize PCBs to help us judge the status of processes. But it is still desirable to provide the capability of monitoring the status of processes within the operating system. In particular the fact a process has been terminated or abnormal usage of CPU and memory space is detected.

1.2. Process Configuration

In this section, we will introduce useful terms for process monitoring as follows [2]:
(1) PID (Process ID): Every process is assigned a unique identifier by operating system when it is created, and it is used by operating system to physically locate the process. User can easily obtain a process list. (using Task Manager on Windows or ps on UNIX)
(2) Process Name: Every process has non-unique, symbolic name, which is a variable-length string of characters.
(3) CPU Usage: It shows the percentage of elapsed time that all of the threads in this process used to execute instructions.
(4) Memory Usage: For Windows monitoring, this parameter shows the current number of megabytes in the working set of this process. The working set is the set of memory pages touched recently by the threads in the process.

2. BACKGROUND

2.1. System Framework and Components

The framework of the LCD defect inspection system consists of the 9 CCD (Charge-Coupled Device) cameras, 36 industrial personal computers, and 2 central computers.

The LCD defect inspection system works as follows. First of all, the analog image signal output from each CCD camera is transferred to the 4 industrial personal computers via Camera Link. Then the frame grabber of each industrial personal computer performs an analog-to-digital signal conversion and put digitalized image into a pre-allocated memory buffer. The frame grabber also performs fast image matching (fast Fourier transform, image subtraction with golden images). Finally, the main computer categorizes the defects and report details of the defects on the review station.

The framework of the LCD defect inspection system is illustrated in Fig. 1.

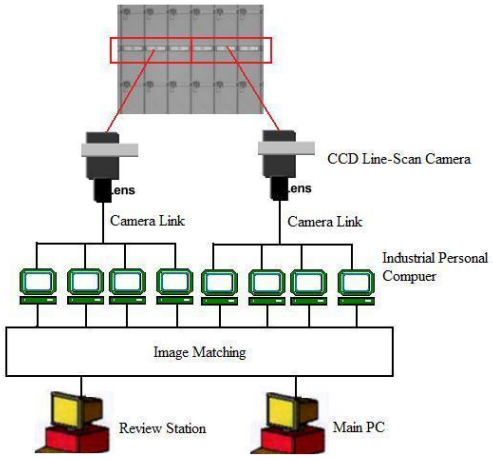

Fig. 1: The framework of the LCD defect inspection system

2.3. Remote Procedure Call

Remote Procedure Call (RPC) is a network programming standard originally developed in the early 1980s [8]. The Open Software Foundation (OSF, now The Open Group) made RPC part of the Distributed Computing Environment (DCE) distributed computing standard. Although there is a second RPC standard, SunRPC, the Microsoft RPC implementation is compatible with the OSF/DCE standard. RPC builds on other networking APIs, such as named pipes or Winsock, to provide an alternate programming model that in some sense hides the details of networking programming from an application developer.

RPC applications work like this: As an application runs, it calls local procedures as well as procedures that are not present on the local machine. To handle the latter case, the application is linked to a local static-link library or Dynamic Link Library (DLL) that contains stub procedures, one for each remote procedure. For simple applications, the stub procedures are statically linked with the application, but for bigger components the stubs are included in separate DLLs.

2.3. Related Work

In this section we will list all materials related to our research. M. E. Russinovich et al. [8] describe each Windows process represented by an executive process (EPROCESS) block. Fig. 2 shows the data structure of an executive process block. M. E. Russinovich et al. also introduce how Microsoft Windows builds its memory management mechanism and how the mechanism works. G. Schmidt [9] introduces how to measure CPU usage of processes and threads and uses Windows system API (Application Programming Interface) to implement it. Helsper et al. [3] propose a method and system for computing a performance forecast for an e-business system or other computer architecture to proactively manage the system to prevent system failure or slow response time. Richardson [7] introduces the algorithm and many flowcharts to monitor health problems of network devices and services of a managed network environments. F. Jahanian et al. [5] introduce runtime monitoring of timing constraint in distributed real-time systems using Graph Theory lemma and algorithms.

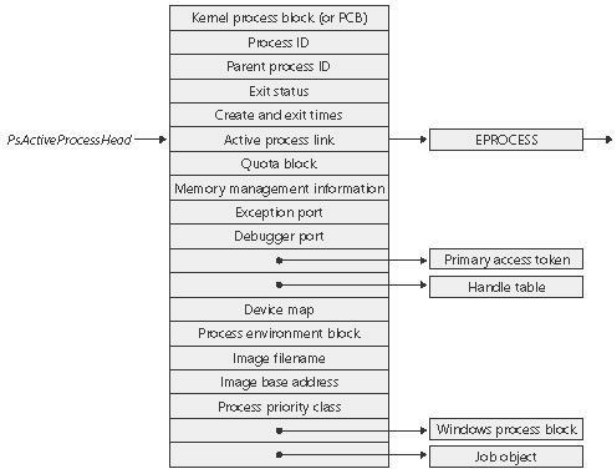

Fig. 2: Structure of an executive process block

2.4. Case Studies

In this section, we will introduce some critical software mainly on the Windows platform for process monitoring, and we also point out its useful functions in detail.

2.4.1 PerfMon

We can observe the following three counters to give a good overview of general activity of our system [1]:
(1) Process utilization:
Processor\% Processor Time_Total - Just a handy idea of how 'loaded' the CPU is at any given time. Do not confuse 100% processor utilization with a slow system though - processor queue length, mentioned above, is much better at determining this.
(2) Memory utilization:
Process\Working Set_Total (or per specific process) - This basically shows how much memory is in the working set, or currently allocated RAM.
Memory\Available MBytes - Amount of free RAM available to be used by new processes.
(3) Network utilization:
Network Interface\Bytes Total/Sec\nic name - Measures the number of bytes sent or received.

2.4.2 JP1

JP1 is the software of the integrated systems management developed by Hitachi [4]. It offers job management, desktop management, and integrated management for business user. It has the following critical functions:
(1) Server downtime monitoring:
This is an important step to ensure that the system is running. We might take that for granted, but what a disaster it would be if the system had stopped without our knowing. This is the simple task of monitoring for a server-down condition.
(2) Performance monitoring:
It includes two parts: response monitoring and cause analysis.
The system might be up and running, but it is hard to know that if jobs are being held up. Delay is often a precursor of failure so Part 1 predicatively detects slower response times.
Cause analysis helps users identify which of the potential causes actually led to the system slowing down.
(3) **Monitoring for future needs:**
The operating data accumulated each day help user gauge the system throughput, and we can use these data for forecasting our future needs for system expansion.

3. PROPOSED METHOD

3.1. Client-Server Architecture

Fig. 3 is our client-server architecture. In this research we need a personal computer as a server computer and other computers as client. The server monitors the clients on Local Area Network (LAN) through the Internet.

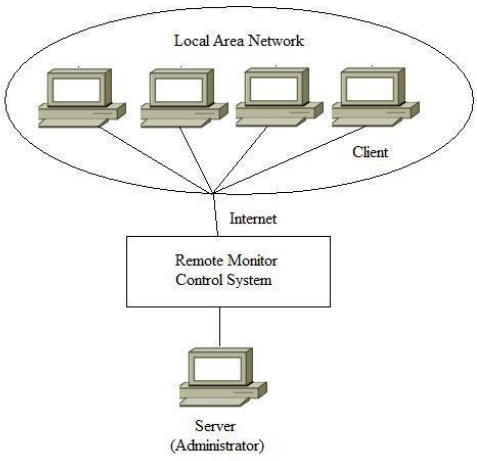

Fig. 3: Our client-server architecture

In this paper, the expected functions our system can provide:
1. Process Monitoring through the Internet by detecting CPU usage, memory usage, and total connections from clients.
2. Administrator can get detail information such as disk and memory space of the clients
3. Administrator can control the client by remote procedure call to reboot or logout.
4. Administrator can remotely lock the task manager of the client in order to prevent intrusion by unwanted external users.
5. If any client violates our criteria, then the server will receive a warning message.

3.2. Function Implementation

3.2.1 Remote Function Call (RFC) 1757

RFC 1757 defines a portion of the Management Information Base (MIB) for use with network management protocols in TCP/IP-based Internets [11]. In particular, it defines objects for managing remote network monitoring devices.

There are many goals for remote network management, the following are goals related to our work:

(1) Proactive Monitoring:

Given the resources available on the monitor, it is potentially helpful to continuously run diagnostics and to log network performance. Our system provides the function of monitoring resources of each client such as file system, disk space, and memory space, and so on.
(2) Problem Detection and Reporting:

The monitor can be configured to recognize conditions, most notably error conditions, and continuously to check for them. When one of these conditions occurs, the event may be logged, and management stations may be notified in a number of ways. Our system provides the criteria to judge whether

the client is out of memory or disconnected, and then sends a warning message to administrator automatically.
(3) Multiple Managers:

An organization may have multiple management stations for different units of the organization, for different functions, and in an attempt to provide disaster recovery. Our system provides three types of protected mode to restrict unwanted user to operate the client computer: Lock/Unlock Desktop, Lock/Unlock Task Manger, and Lock/Unlock Start Button.

3.2.2 Network Monitoring Table

We define two tables for server and client in order to record some critical information which can help us detect and trace the status of client. Table 1 is the record structure for client and Table 2 is the record structure for server.

Record Name	Record Description	Unit
IP	IP address of client	String
PID	Process ID	Integer
pName	Process name	String
CPU_Cur	Current CPU usage	Percentage
CPU_Max	Maximum CPU usage	Percentage
CPU_Min	Minimum CPU usage	Percentage
CPU_Avg	Average CPU usage	Percentage
Mem_Cur	Current Memory usage	Mega bytes
Mem_Max	Maximum Memory usage	Mega bytes
Mem_Min	Minimum Memory usage	Mega bytes
Mem_Avg	Average Memory usage	Mega bytes
CPU_Vote1	CPU votes per inspection period	Integer
CPU_Vote2	CPU votes per inspection period	Integer
CPU_Accu	CPU usage accumulation per inspection period	Integer
Mem_Accu	Memory usage accumulation per inspection period	Float
connTime	Total time client connected to server	Second
sampleCount	When server queries the client everytime then increases this count	Integer
CPU_Table	Whenever we get a CPU_Cur then we add it to this table.	Float
Memory_Table	Whenever we get a Memory_Cur then we add it to this table.	Integer

Table 1: Record structure for client

Record Name	Record Description	Unit
IP	IP address of server	String
connTable	Record all the IP addresses connected to server	String
inspectionPeriod	Total inspection time	Second
totalConn	Total connections of server from clients	Integer
curConn	Current connections of server from clients	Integer
pollingFreq	Query each client every pollingFreq	Second
CPU_Threshold	Threshold of CPU usage	Percentage
CPU_StandardDev	Standard Deviation of CPU usage	Float
Mem_Threshold	Threshold of memory usage	Percentage
Mem_StandardDev	Standard Deviation of memory usage	Float
focusOn	The specific process which is focused by administrator and filters out other processes	String

Table 2: Record structure for client

3.3. Our Proposed Method

3.3.1 Simplified Flowchart of Our Method

First, server will gather data from all clients, and uses the collected data to determine whether any client violates our rule. If yes, then system will send an alarm to administrator and make some restored operations. Fig. 4 shows the simplified flowchart of our method.

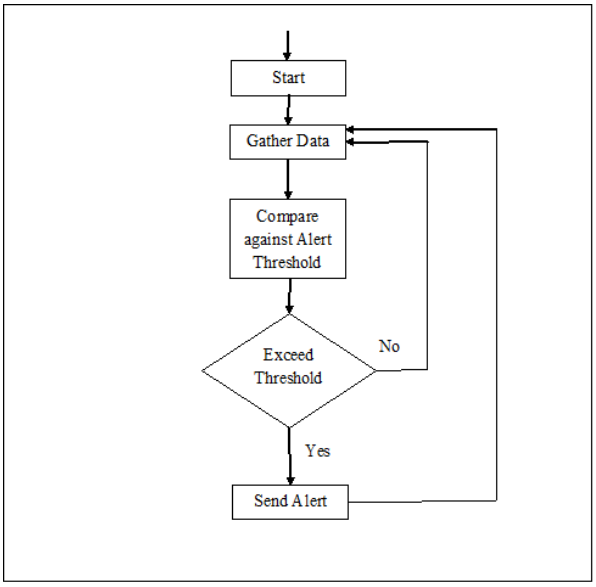

Fig. 4: Simplified flowchart of our proposed method

3.3.2 Training Stage

We will recommend administrator to run this stage if it needs to monitor the specific process in order to get the statistical data to set up more precise parameters such as *CPU_Threshold* and *Mem_Threshold*.
First we accumulate CPU and memory of the specific process on each client every *pollingFreq* seconds in Equations 3.1 and 3.2 (Default = 2 seconds if administrator does not specify).

$$CPU\_Accu = \sum_{t=-pollingFreq}^{inspectionPeriod} CPU\_Cur_{t+pollingFreq} \quad (3.1)$$

$$Mem\_Accu = \sum_{t=-pollingFreq}^{inspectionPeriod} Mem\_Cur_{t+pollingFreq} \quad (3.2)$$

where *CPU_Accu* is the accumulation of the process CPU usage; *Mem_Accu* is the accumulation of the process memory usage; *t* is the current time; CPU_Accu_t is the current CPU usage at time *t*; Mem_Accu_t is the current memory usage at time *t*; and *pollingFreq* is the polling frequency.

We record the maximum and minimum CPU and memory usage as the statistical data as the references to the administrator. Finally we calculate the standard deviation of the process on each client then take average of overall standard deviations (CPU and memory) from all clients as *CPU_StandardDev* and *Mem_StandardDev* in Equations 3.8 and 3.9.

$$CPU\_avg = \frac{CPU\_Accu}{sampleCount} \quad (3.3)$$

$$Mem\_avg = \frac{Mem\_Accu}{sampleCount} \quad (3.4)$$

$$N = \frac{inspectionPeriod}{pollingFreq} \quad (3.5)$$

$$\sigma\_CPU_i = \sqrt{\frac{\sum_{j=1}^{N}(CPU\_Cur_j - CPU\_Avg)^2}{N}} \quad (3.6)$$

$$\sigma\_Mem_i = \sqrt{\frac{\sum_{j=1}^{N}(Mem\_Cur_j - Mem\_Avg)^2}{N}} \quad (3.7)$$

$$CPU\_StandardDev = \frac{\sum_{i=1}^{totalConn} \sigma\_CPU_i}{totalConn} \quad (3.8)$$

$$Mem\_StandardDev = \frac{\sum_{i=1}^{totalConn} \sigma\_Mem_i}{totalConn} \quad (3.9)$$

where *CPU_avg* is the average of the process CPU usage; *Mem_avg* is the average of the process memory usage; *N* is the total sample counts; *i* is the index of the client; *totalConn* is the total clients connected to server; σ_CPU_i is the CPU standard deviation of CPU usage the process running on client *i*; σ_Mem_i is the standard deviation of memory usage of the process running on client *i*; *CPU_StandardDev* is the average CPU standard deviation of all clients; and *Mem_StandardDev* is the average memory standard deviation of all clients.

3.3.2 Determining Stage

In this section, we will divide three parts to diagnose the status of the process: CPU, memory, and network connections.

(1) CPU aspect:
We compute the average for CPU usage of the specific process running on each client every *pollingFreq* second (Defalut *pollingFreq* = 2 seconds). We use Equations 3.10 and 3.11 to estimate the frequency whether the CPU usage is higher or lower than the boundaries.

$$CPU\_Votes1 = \begin{cases} CPU\_Votes1 + 1, if CPU\_Cur > CPU\_Avg + CPU\_StandardDev \\ CPU\_Votes1, otherwise \end{cases}$$

(3.10)

$$CPU\_Votes2 = \begin{cases} CPU\_Votes2 + 1, if CPU\_Cur < CPU\_Avg - CPU\_StandardDev \\ CPU\_Votes2, otherwise \end{cases}$$

(3.11)

We use the threshold defined by the administrator to determine whether the frequency is too high. If it is, then we send an alarm to administrator.

(2) Memory aspect:
Here we have two approaches for making the decision. One is similar to CPU aspect (1), but in common monitoring software for memory monitoring, it usually uses the following method.
We set the upper bound and lower bound for memory space monitoring. If the current memory usage of the process is higher than *Mem_Max* or is lower than *Mem_Min* then we send an alarm to administrator.

(3) Connection aspect:
Whenever a client connects to server, server saves the IP address of the client into *connTable*. Sometimes client will disconnect or logout so we create the temporarily connecting table to save the current IP addresses of clients on live processes. We can check whether *totalConn* and *curConn* are the same, if not we compare *connTable* and temporarily connecting table to find out which IP address is missing and send an alarm to administrator.

4. EXPERIMENTAL RESULTS

4.1 Experiment Environment

First, we introduce our experiment environment:

Server:
CPU: AMD Athlon64 3200+ 2GHz
Memory: 3GB
OS: Windows XP
Programming Language: Microsoft Visual C++ 2005 with Microsoft Foundation Class (MFC)
Client:
We pick 2-4 computers at random which are grouped on a local network.

4.2 Test Condition Setup

We set up three conditions for testing memory usage, and network status monitoring:
(1) Memory usage:
We choose four computers as client running the Matlab application and pick two of them executing the duplication. The following are configurations of this experiment.

Number of clients: 4
Running application: Matlab executing Weighted Least Square filter program.
Inspection period: 5 minutes
Memory statistical data after training stage:
Mem_Min: 162 MB
Mem_Avg: 195 MB
Mem_Max: 362 MB

(2) Network connection:
We choose four computers as client and manually disconnect three of them as the experiment.

Total clients: 4
Number of clients disconnected manually: 3

4.3 Precision and Recall

Precision and recall are two widely used measures to evaluate the quality of results in monitoring system. In our experiment, *Found* is the true error detected by our algorithm; *Total* is the ground truth; and *Correct* is the correct warning by our algorithm. Fig. 5 shows their relationship.

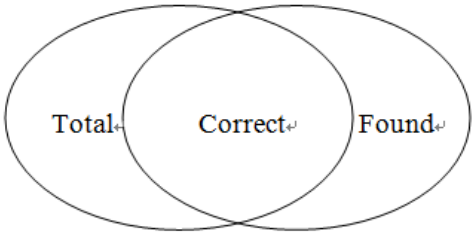

Fig. 5: Total, Correct, and Found relationship

Precision and recall are calculated as follows:

$$Precision = \frac{Correct}{Found} \text{x}100\% \quad (4.1)$$

$$Recall = \frac{Correct}{Total} \text{x}100\% \quad (4.2)$$

Our experimental precision and recall table is in Table 3.

Type	Found	Total	Correct	Precision	Recall
Memory	2	2	2	100%	100%
Memory	1	1	1	100%	100%
Network	3	3	3	100%	100%
Network	2	2	2	100%	100%

Table 3: Precision and recall comparison table

It is hard to define what *Found* and *Total* in CPU usage monitoring, but our system can successfully detect the abnormal CPU usage when the client executes other external applications with heavy CPU loaded.

The following is the comparisons between our system and Windows Perfmon. Our system has the same precision and recall but our system has many more functions.

Functionality	Our System	Perfmon
Auto Alerting	Our system supports three types of alerting included CPU, memory, and the network connection after the determining stage.	No support, the administrator must make the decision by himself (herself) after observing the system status plotting chart.
User Interface	Clear, and easy to use even though the administrator has no computer background.	The user interface is very complete, but it has so many configurations to be set. It is a little hard to begin starting.
Plotting Scheme	Not supported.	Provide the complete plotting scheme as the references for administrator.
Accuracy	Perform well on memory and network monitoring.	It has no auto alerting scheme, so we do not know how accuracy it is.
Remote Control	Integrated the remote control functions into our system.	Must use the external applications to control the clients.

Table 4: Comparisons between our system and Microsoft Perfmon

5. CONCLUSIONS AND FUTURE WORK

5.1. Conclusion

Our system compared with Microsoft perfMon is simple and easy to use even for the administrator without computer background. It offers a friendly graphic user interface for the administrator to remotely control and observe the client computer when needed. It also provides the protective mechanism such as Lock/Unlock the Task Manager to prevent unwanted users control the client computer.

Although our decision algorithms of this system are not robust to handle many varieties of problems on the actual network environment, our system works well in most environments. The quality of our system compared with the commercial suites is slightly poorer because we do not support as many detailed parameters due to short implementation time.

5.2. Future Work

Our system is a client-server architecture based on Remote Procedure Calls. RPC is required which could cause problems with firewalls or anti-virus software. In the future we will modify this part in order to remote control the client more safely.

The other issue is to improve our algorithms to increase the accuracy of detecting troubles and let our algorithms have enough strength to handle many complex conditions.

Finally we want to expand our system to be cross-platform such as hand-held system. That is to say the administrators may not always sit in front of the server computer to operate the client computers remotely, administrators can control by cell phones or other hand-held device instead.

REFERENCES

[1] Adminfo.com, "Windows Perfmon: The Top Ten Counters," http://adminfoo.net/2007/04/windows-perfmon-top-ten-counters.html, 2007.

[2] DisplaySearch, "Large-Area TFT LCD December Revenues Fall to $3.0 Billion, Reaching the Lowest Level in Two Years; Samsung #1 in Revenues and Korean Manufacturers' Unit Market Share Continues to Exceed 50%," http://www.displaysearch.com/cps/rde/xchg/displaysearch/hs.xsl/01_22_2009_large_area_tft_lcd_december_revenues_fall_to_3_billion.asp, 2009.

[3] D. Helsper, C. Wilkinson, R. Zack, J. T. Tatum, R. J. Jannarone, and B. Harzog, "Enhanced Computer Performance Forecasting System," United States Patent #6876988, http://www.google.com/patents?id=SesVAAAAEBAJ&dq=6876988, 2005.

[4] Hitachi, "JP1," http://www.hitachi-eu.com/jp1/solutions/index.jsp, 2009.

[5] F. Jahanian, R. Rajkumar, and S. C. V. Raju, "Runtime Monitoring of Timing Constraints in Distributed Real-Time Systems," *Real-Time Systems*, Vol. 7, No. 3, pp. 247-273, 1994.

[6] Patrol Express, "Memory Usage Parameter," http://patrolexpress.bmc.com/help/en_US/gal_webhelp/parameters/process/pr_memory_usage.htm, 2009.

[7] D. E. Richarson, "Dynamically Drilling-Down through a Health Monitoring Map to Determine the Health Status and Cause of Health Problems Associated with Network Objects of a Managed Network Environment," United

States Patent #7146568, http://www.google.com/patents?id=uht9AAAAEBAJ&dq=7146568, 2006.

[8] M. E. Russinovich, *Microsoft Windows Internals*, 4th Edition, Microsoft Press, Redmond, WA, 2004.

[9] G. Schmidt, "How to Get CPU Usage of Processes and Threads," http://www.codeproject.com/KB/system/processescpuusage.aspx, 2009.

[10] G. Simor, "Process Creation and Termination Monitors for Use in a Distributed Message-Based Operating System," United States Patent #5060150, http://www.google.com/patents?id=1RQXAAAAEBAJ&dq=Process+Creation+and+Termination+Monitors+for+Use+in+A+Distributed+Message-Based+Operating+System, 1991.

[11] S. Waldbusser, "RFC 1757," http://www.faqs.org/rfcs/rfc1757.html, 1995.