U.P.B. Sci. Bull., Series C, Vol. 73, Iss. 4, 2011 ISSN 1454-234x

ARCHITECURE OF THE LINUX WATCHDOG SOFTWARE FOR THE IP PBX

Mihai Constantin[1]

În această lucrare sunt proiectate propunerile de arhitectură referitoare la software-ul de Watchdog pentru monitorizarea unui IP PBX. IP PBX conține două aplicații principale care interacționează între ele: Common Media Server şi IpSmallOfice. Common Media Server oferă capabilitățile de codec VoIP, iar IPSmallOffice, aplicația principală care rulează pe Linux PC, controlează apelurile telefonice şi oferă telefonie IP. Watchdog reprezintă o aplicație de sine stătătoare şi necesită un fişier de configurare care să citească date despre procesele pe care le monitorizează. De asemenea, această aplicație rulează ca „daemon" şi este pornită la startup-ul întregului sistem IP PBX.

The architecture proposals of a Watchdog software for the IP PBX monitoring are designed in this paper. IP PBX contains two main applications which interact between them: ***Common Media Server and IpSmallOffice. Common Media Server*** *provides VoIP codec capabilities, and* ***IpSmallOffice*** *is the main application running on a Linux PC, controls the phone calls and provides IP telephony switching. The Watchdog is a stand-alone application and it needs a configuration file to read the data about the processes it monitors. Moreover, this application will be demonized and is added to start at system startup.*

Keywords: Watchdog, Common Media Server, IpSmallOffice,Linux, Processes,Polling,SIP,Keepalives

1. Introduction

IPSmallOffice and Common Media Server are interdependent applications. Given this, when one of them is shutdown or, in a worse case, crashes, both of them need to be restarted. This is where the watchdog comes in and helps ensure the recovery of functionality in the best conditions.

IPSmallOffice and Media Server are communicating using the configured number of pre-allocated SIP connections established at startup by IPSmallOffice. However, every time the IPSmallOffice boots it establishes a new set of SIP connections. Even though the first set of established connections is no longer used, Common Media Server keeps them for a given amount a time. If there is a

[1] PhD Student, Faculty of Automatic Control and Computer Science, University POLTEHNICA of Bucharest, Romania, e-mail: mconstantin@luxoft.com

potential crash of the IPSmallOffice application however, these connections are not cleaned and Common Media Server should be restarted.

However, given that IPSmallOffice is a process in Linux, if there should unfortunately happen to be a segmentation fault, the process will core and exit, but it will not be started again automatically. This issue could affect the operability of the system.

There could be also the reverse situation when Common Media Server is the application that exists unexpectedly, that is if any of the four component processes terminates unexpectedly.

2. Related work: Similar solutions

A watchdog timer is associated in the literature with two approaches: a computer hardware or a software timer that triggers a system reset or other corrective action if the main program, due to some fault condition, such as a hang, neglects to regularly service the watchdog (writing a "service pulse" to it). The intention is to bring the system back from the unresponsive state into normal operation.

Watchdog timers can be more complex, attempting to save debug information onto a persistent medium; i.e. information useful for debugging the problem that caused the fault. In this case a second, simpler, watchdog timer ensures that if the first watchdog timer does not report completion of its information saving task within a certain amount of time, the system will reset with or without the information saved. The most common use of watchdog timers is in embedded systems, where this specialized timer is often a built-in unit of a microcontroller.

Even more complex watchdog timers may be used to run unsafe code in a computer security.

Watchdog timers may also trigger fail-safe control systems to move into a safety state, such as turning off motors, high-voltage electrical outputs, and other potentially dangerous subsystems until the fault is cleared.

For those embedded systems that can't be constantly watched by a human, watchdog timers may be the solution. For example, most embedded systems need to be self-reliant, and it's not usually possible to wait for someone to reboot them if the software hangs. Some embedded designs, such as space probes, are simply not accessible to human operators. If their software ever hangs, such systems are permanently disabled. In cases similar to these, a watchdog timer can help in solving the problem.

The watchdog timer is a chip external to the processor. However, it could also be included within the same chip as the CPU; this is done in many microcontrollers. In either case, the watchdog timer is tied directly to the processor's reset signal. Expansion card based watchdog timers exist and can be fitted to computers without an onboard watchdog.

Most of the watchdogs are found on different PBX platforms as hardware watchdog. This watchdog needs to be ping-ed every few seconds to ensure the monitored application is working correctly. If this does not happen, it reboots the PBX. This type of implementation isn't blocked waiting for a resource or other, as the watchdog resets the PBX.

The watchdog solution discussed on this paper is not a hardware one. On Linux, there already are two types of watchdogs. There is a hardware one and also a software one: softdog. Also, in the process of deciding the best solution, the preemptive Linux Kernel is considered.

Here are some research and comparisons with other similar process monitoring and control tools.

The watchdog will be started, as all the other processes, at system startup and will be running until reboot/shutdown of system. The watchdog will have a configuration file that specifies all the information about the processes to monitor.

The configuration file about the processes contains also the PID of the applications.

When a certain application has exited, the PID and binary is no longer in /proc. The Watchdog will first check if it was a crash or a clean shutdown by looking for the lock file, then, if it was a crash, it will restart the application and scan for the new PID.

Another tool for managing and monitoring processes is Monit. It is a utility for managing and monitoring processes, files, directories and devices on a Unix system. Watchdog architecture for IP PBX manages only two main applications: Common Media Server and IpSmallOffice.

The Monit, also, needs a configuration file to read data monitoring processes, both (Watchdog and Monit) are demonized and start at startup. The Watchdog configuration file contains less information than the Monit.

The Watchdog needs the TinyXML parser to extract the information, because configuration file format is XML that makes the interaction between the Watchdog and the processes more easily for understanding and control.

Another existing tool for monitoring processes and tasks is God. This can be configured easily because the config file is written in Ruby.

Example of configuration file :

```
                  God.watch do |w|
                  w.name = "gravatar2-mongrel-#{port}"
                  w.interval = 30.seconds # default
                  w.start = "mongrel_rails start -c #{RAILS_ROOT}
-p #{port} \
                              -P #{RAILS_ROOT}/log/mongrel.#{port}.pid
-d"
                  w.stop = "mongrel_rails stop -P

                        #{RAILS_ROOT}/log/mongrel.#{port}.pid"
                  w.restart = "mongrel_rails restart -P

                        #{RAILS_ROOT}/log/mongrel.#{port}.pid"
                  w.start_grace = 10.seconds
                  w.restart_grace = 10.seconds
                  w.pid_file = File.join(RAILS_ROOT,
"log/mongrel.#{port}.pid")

                  ...
                  end
```

To start the God monitoring process as a daemon simply run the god executable passing in the path to the config file.

There are two methods of monitoring all the applications in the Watchdog: polling and keepalives. The default method of monitoring for all added applications is polling for the watchdog. When the Watchdog is monitored through keepalives, no message is received and the lock file is still present.

The Monit is more complex than Watchdog because it manages also files, directories and devices. Features included in the Monit are to start, stop or restart services and processes, dependencies between services configuration. Also, the Monitor can be configured to send alert notifications and alert reports on service error and recovery.

God tool uses only polling as method of monitoring.

The Watchdog deals with situations where one application fails. It can recognize and differentiate crash and a clean shutdown.

The Watchdog configuration file contains the general parameters for the watchdog behavior, the polling interval and the list of processes. An application has a specific information: process name, start command, the path of the file used to monitor start/stop, starting delay , the value of keepalive if the application is monitored through keepalives.

The Monit configuration file contains the pooling interval and the list of processes and services. A process in Monit configuration file contains specific information such as from the Watchdog configuration file: start and stop command , the path of the file used to monitor (pidfile). Moreover, the Monit configuration file contains address where it sends the alert messages. Also Monit can test the identification number of a process. This test is useful to detect possible process restarting which has occurred between two testing cycles.

The Wachdog, Monit and God are similar solutions used for monitoring and controlling processes and can manage the behavior of the entities controlled.

3. Requirements

Given the problems presented above, a common solution that should tackle all of them is required – watchdog application.

The watchdog handles the following situations which are analyzed below.

Media Server is up, IPSmalloffice has crashed. The watchdog detects the IPSmalloffice crash, close Media Server and start them both as a normal start operation.

Common Media Server is up, IPSmalloffice has rebooted. Common Media Server connections are already handled from the IPSmalloffice restart code. Watchdog cannot detect this behavior as the restart is implemented as exec on self.

Common Media Server is up, IPSmallOffice has closed manually (not likely, but should be considered). This situation mainly can appear only if the admin-user manually issues a "service ipoffice stop" command in the Linux console. The proposal is that watchdog also stops Common Media Server, as there is no reason to keep Common Media Server running with the pre-allocated connections when IPSmallOffice was stopped.

Common Media Server has closed manually, IPSmalloffice is up. This is the same situation as above.

Common Media Server has crashed, IPSmalloffice is up. The watchdog detects the Common Media Server crash, closes IPSmalloffice and starts them both as normal start operation. Although IPSmallOffice has voice path without Common Media Server using direct media, when Common Media Server gets up again, IPSmallOffice does not detect it. This is why they both are restarted. If IPSmallOffice could re-establish connections to the newly restarted Common Media Server, then only Common Media Server is handled.

However, the watchdog differentiates between a crash of the application and a clean shutdown. If the application was shutdown manually, it isn't restarted automatically.

Watchdog is easily extendable to cover other applications that might need monitoring in the future.

Watchdog is a robust application, so to ensure that it does not crash under stress conditions and thus monitoring will fail.

As a failsafe, watchdog is re-spawned from /etc/inittab file, to ensure functionality, even if it happens to crash. This re-spawn behavior might have further implications that should be considered if this requirement is added. These implications depend also on the solution preferred.

There is some logging set in place to keep track of the actions performed by the Watchdog: start/stop of processes, crashes detected, etc, as this is useful for debugging the applications that were crashed and restarted.

The Watchdog has additional requirement to check hard disk space availability and if this reaches a certain limit, it should check for cores and do cleanup so that cores do not take a high percentage of the disk space.

If the watchdog is able to monitor other applications, there is a configuration file that specifies the applications to monitor, their binary name and path, their start/stop commands and parameters and also some solution-specific values (polling interval, etc).

There is a possibility that IPSmallOffice gets into a state where it does not function correctly (not responding or other) but the process is still running. This type of behavior is either because it is waiting for a resource to be freed, or just some intensive processing is happening at the moment. In this case the watchdog cannot detect the problem.

Another very important issue is that watchdog detects between IPSmallOffice being properly shutdown and being crashed. This cannot be done, however, based on the current way the IPSmallOffice is functioning. It is necessary to handle the correct shutting down and notify the Watchdog about it.. Also, if the design allows it, the use of waitpid helps getting the exit status for these processes and the crash is detected this way.

4. Design proposals. Contributions

Based on the research, the following solutions are designed and they are the main contributions of this paper. These contributions are highlighted on this section.

4.1. Solution 1 – Keepalives

A first contribution is by designing a Watchdog application that needs pinging every few seconds. In case the pings are lost, a timer expires and resets the IPSmalloffice/Common Media Server applications.

The Watchdog is a daemon that starts using one of the options in 4.4 – Start/Stop mechanism. This daemon runs a timer set to 10 seconds (this period should be set so to avoid extra processing but to ensure a time to return to service short enough). The daemon is "listening" to IPSmallOffice using a message queue where IPSmallOffice sends periodically a message to the application: APPS_KEEPALIVE. Every time this message is received, the daemon resets the IPSmallOffice timer. When no message is received, given that the timer was no longer reset, it will expire after the set period and the callback method will restart both IPSmallOffice and Common Media Server.

There is a certain problem that needs be addressed. This is the reset by hand of the IPSmallOffice. In this case, the watchdog knows that the absence of messages is due to user actions. The solution is implemented by adding two more messages: APPS_STOPPED and APPS_STARTING to let the watchdog know the state of the IPSmallOffice is user controlled and not due to a crash.

Another requirement is the possibility that Watchdog to be re-spawned by using init. In this case, the only issue might that, when Watchdog restarts, it should know the state the applications were when it was restarted. This state is written somewhere every time it changes for each application so that at restart it is loaded and Watchdog can continue from exactly the same point it was closed.

The advantage is that this solution relies more on the IPSmallOffice to offer information and keeps Watchdog simpler.

The disadvantage is the overridden if this call is added whenever there is a task change, but this might happen at a 30 seconds interval, so it has to support this kind of delay in processing the messages.

4.2. Solution 2 – Polling

Another contribution is by designing a solution that starts from the concept of having as little interaction and changes in IPSmallOffice/Common Media Server. The main idea behind the solution is to have an application that runs in the background and that is polling the processes and takes action according to their status.

The Watchdog is a daemon that starts, similar to solution 1. It has a continuous loop that should only be interrupted at exit. For every iteration, the watchdog checks the status of the IPSmallOffice/Common Media Server. If they are all up and running, the Watchdog does a sleep for 20 seconds – this value is according to the one used in solution 1, to ensure an optimal time to return to service, while not doing extra processing. In this solution, the Watchdog cannot wait for the process exit status, as the process is not a child. If any of the checked processes is down, it checks the way the process has been exited: whether it crashed or exited nicely – user action. Depending of the way the process exited, the Watchdog restarts the process or leave it as it is. After all the appropriate actions have been taken, the watchdog sleeps the number of seconds established. This solution handles in the same way all the applications monitored.

It is also addressed here the re-spawn requirement, related to this solution. Given that the polling is done by PID, this is saved every time a process is started and removed when it is stopped. When Watchdog restarts, it loads the list of PIDs, checks it against the existing PIDs and takes appropriate action.

The main advantage is that all the monitoring is done inside the watchdog, with small changes in the applications, this approach being innovative in some aspects. Another advantage is that the only changes needed in the IPSmallOffice are a mechanism to check if process exited cleanly or it crashed. This can be solved by looking at the process exit code, if it is correctly used in the code and if the processes are children processes – see solution 3 for this.

The disadvantage is that this solution does not cover the blocking of the process.

4.3. Solution 3 – Complex control

This solution starts from the idea (main contribution) of giving Watchdog the responsibility of starting and stopping IpSmallOffice/Common Media Server and having the possibility of monitoring them as children processes.

Watchdog is a daemon that starts up along with the OS. It has a lock that ensures there is only one instance running at a time. The watchdog is running all the time the OS is up. The IPC mechanism used is message queues. When the watchdog starts up, it sets the lock and opens a message queue to receive start/stop commands. When the user wants to start/stop an application, he should run the command "watchdog start <application_name>" or "watchdog stop <application_name>". When this new watchdog starts, checks the lock, does see the lock is taken and another watchdog is already up and monitoring. After that, it checks the arguments received and sends messages to tell the monitoring watchdog what actions to take, and then it will exit.

When Watchdog starts the applications it will have the PIDs for them, it can wait for them to terminate and it can get the exit status they had. Using this exit status, it can see if the process has terminated cleanly or if it has crashed, hence having the possibility to take the appropriate actions: restart IpSmallOffice/Common Media Server.

The main problem with this solution is that Common Media Server is, by design, started using a script with the command "service acms start". The script does some checks and configuring and after all is set up, starts the four processes that form Common Media Server.

Given this behavior there are two possibilities.

This first one is to reproduce from code all that is happening in the acms script for both start and stop, which is rather complicated and adds to the complexity of the solution all together. Another issue with this option is that it also implies maintenance work at every Common Media Server release to ensure that all changes performed on Common Media Server script will reflect in the code.

The second possibility is to start Common Media Server just by running "service acms start" using the script, but this way it cannot monitor the state of Common Media Server process with waitpid.

Both ways to fix these architecture issues are original contributions.

There are implications for this solution from the re-spawn requirement. Given that this solution handles children processes, if the watchdog is restarted, the children will have as parent the init process. When watchdog starts, it will not have access to the wait methods for the initial children. The solution for this is that Watchdog maintains a list of the children PIDs and with the use of an “Adoption” kernel module that should handle the process table, the newly started Watchdog can adopt the children that were passed to init.

The advantages are**:** watchdog can control and monitor the applications better, also watchdog is easily extendable to handle other applications.

The main disadvantage is there is a single point of failure, if watchdog crashes, because the applications are started through it. Also if the exit status is not set appropriate every time exit is called, it might be considered a inconsistent behavior.

4.4. Start/Stop mechanism

The start/stop mechanism for Watchdog is closely related to that of the IPSmallOffice in that watchdog should always be informed when the IPSmallOffice is closing gracefully. Whatever options can be found for managing watchdog, there still has to be implemented in the IPSmallOffice the handling of SIGTERM/SIGKILL so that IPSmallOffice exits gracefully and could notify the watchdog of this.

There are several ways the watchdog can be integrated into the start/stop mechanism, and they are analyzed below as original contributions.

The watchdog daemon can start at boot and run while the OS is running. This possible solution relies on the communication with IPSmallOffice. IPSmallOffice notifies the Watchdog when it starts up, at which moment the Watchdog starts its monitoring behavior. When IPSmallOffice is shutdown cleanly, it notifies the Watchdog which will exit the monitoring until notified again that IPSmallOffice is running. The advantage is that is the most generic solution, as whatever other application monitorized with the Watchdog, it can be easily covered, without changing the start/stop of the daemon. The disadvantage is this solution relies on notifications for the proper monitoring behavior. The notification system should be very reliable.

Watchdog is started by the IPSmallOffice start/stop scripts, right after Common Media Server and IPSmallOffice started and isn’t stopped until the OS reboots. When IPSmallOffice is cleanly stopped, it notifies the Watchdog so that it knows what type of shutdown it was. The Watchdog continues monitoring, but it keeps the state of the IPSmallOffice saved so that doesn’t restart it. When IPSmallOffice starts, it will notify the Watchdog again that it is up and the monitoring should not ignore it anymore. The advantage is that even if it only starts with IPSmallOffice’s first run, it will run until the machine is rebooted, so every other application that will start AFTER the IPSmallOffice does can be monitored. The disadvantage is this solution ties the Watchdog behavior and monitoring strictly to the start of IPSmallOffice and Common Media Server.

Another solution is the case when Watchdog uses the same start/stop scripts as the IPSmalloffice and Common Media Server. The Watchdog is started right after Common Media Server and IPSmalloffice are already started. When IPSmalloffice exits cleanly, it will notify the Watchdog. When receiving this notification, the Watchdog exits. The advantage is less use of notifications – only for shutdown. The disadvantage is this solution ties the Watchdog behavior and monitoring strictly to IPSmalloffice and Common Media Server, thus not allowing monitoring other applications, making it less easily extendable**.**

Another way is to use the Watchdog for starting/stopping all the applications. This way it knows all the process ids needed for monitoring. The advantages are there is no need for notifications from one process to another. Watchdog has also more control over the processes in IP PBX. The main disadvantage is that application is more complex, thus more prone to crashes. Another disadvantage is that if exit codes are not reliable, there is need for detection between crash and clean shutdown.

4.5. Differentiate between crash and shutdown

There is an important requirement that Watchdog differentiates between an application crash and a clean shutdown issued either from the web interface or manually from a terminal. In order to achieve this, the solution proposed is a rather simple one, given that all the applications we have are started/stopped using scripts.

When one application is started, the start script should create an application-specific file, where it could optionally write the PID of the started application. When the application is stopped, the stop script should delete the file it has created at start.

After this mechanism is set in place, when the watchdog detects an application has terminated it looks for the application-specific file created by the script at start. If the file is still present, then there was a crash. If the file is deleted and watchdog cannot find it, then the shutdown was user action.

This mechanism is useful to detect and define the situation and restart and reconfigure the processes. It is innovative in some aspect, and as it is not present in other similar solutions(like Monit, God…), this is a main contribution of this paper .

4.6. Common Media Server components and behavior

Common Media Server is formed by several processes. The first one that should be mentioned is initacms, which is started as a daemon. This process, based on a configuration file, starts as children other four necessary processes: acmsComms, msmlCon, acmsdispatcher and aSoftMS. From the investigations made so far, initacms plays the role of watchdog for all his children, restarting it if anything happens to it. So, to ensure Common Media Server not crashing all we have to do is check initacms. If this process goes down, then Common Media Server should be started.

5. Conclusions. New innovative aspects. New contributions

The original contributions are by designing a solution based on a mixture of "Solution 1 – Keepalives" and "Solution 2 – Polling" analyzed above, using also the method from "3.5 Differentiate between crash and shutdown" to handle start/stop and to differentiate between manual stop and crash. The main features of the proposed solution, the keepalives and the polling mechanism, will be designed in order to clarify originality and value of this.

The keepalives will be used for IPSmallOffice monitoring, because its code can be accessed and it is ideal to prevent deadlocks or other similar situations for the applications. The polling will be used for all the other third-party applications which can be monitored.

For the applications where the start script can get the PID (ipsmalloffice, initacms) this value should be written inside the application-specific lock file we use during the lifetime of the process. This will help Watchdog to get the PID without going through the /proc folder to find it. This approach is innovative and by this contribution it is detected the crash of a process. At every polling iteration the watchdog will simply check that the PID and the process it ran are still valid and up.

The IPSmallOffice can be monitored through both methods: polling or keepalives. An option should be added in the Watchdog configuration file so that the monitoring method can be selected. When the Watchdog is monitored through keepalives, no message is received and the lock file is still present, the Watchdog could send a signal to IPSmallOffice to make it generate a core.

The current status of this research is that is found a proper solution, but it needs to do some real experiments with it(by prototyping it).

To understand the importance of a software watchdog, let's look at what many existing systems use to recover from software faults: a hardware watchdog attached to the processor reset line. Typically, a component of the system software checks for the system integrity, and then strobes the timer hardware to indicate that the system is “sane”. The good news is that system recovers from the software or hardware lockup. The bad news is that system must also completely restart, which defeats our goal of high system availability.

Compare this behavior to a software watchdog, which can intelligently choose from several, less drastic, recovery methods. Instead of always forcing a full reset, the software watchdog could:

-simply restart that process without shutting down the rest of the system, or

-abort any related processes, initialize the hardware to a ’safe’ state, and restart the related processes in a coordinated manner or,

-if the failure is critical, perform a coordinated shutdown of the entire system and sound an audible alarm to notify the maintenance staff

The software watchdog lets retain programmed control of the system, even though several processes with the control software may have failed. A hardware watchdog timer can still help recover from hardware “latch-ups”, but for software failures it could have much better control. Furthermore, by employing the “partial restart” approach, the system can survive intermittent software failures without experiencing any downtime.

While performing a partial restart, the system can also collect information about the nature of the software failure. For example, if the system contains or has access to mass storage (flash memory, hard drive, a network link to another computer with a hard drive), the software watchdog can generate a chronologically archived sequence of process dump files. These dump files can then give the information to fix the issues before experiencing similar failures.

A software watchdog not only decreases costly downtime, but also helps to avoid software faults in the future.

Here are analyzed and proposed some innovative aspects regarding the subject of this paper.

A first aspect is the contribution to design a formal language to specify dependencies between monitored and controlled entities. This helps in understanding what happens with entities monitored. The innovative proposal for this is to add in configuration file new information about dependencies. This information can be added in a new attribute in the Watchdog XML configuration file structure.

Example

<**dependson** value=”.... ”/>

The value can be the path to the pidfile of the process or the process name.

Moreover, this new improvement helps the Watchdog to detect easily a crash and deals with it according with the situation. This makes the Watchdog more clear related of what it decides to do in a lot of situations.

One of the most important advantages of this improvement is that the Watchdog can control and monitor better the entities and take the best decisions.

A second innovative aspect that could be designed in the Watchdog architecture is a generic interface for inter-process monitoring and control**.** By this contribution, the interface enables the Watchdog to monitor and control communications between processes.

The approach is based on the communication between the applications and the Watchdog. This communication is made with an unidirectional message queue for the Watchdog that can receive the status of the applications. The design of this aspect can be done with an additional message queue to communicate bi-directional between the applications. Also the Watchdog configuration file needs to be modified.

An application can communicate only with the Watchdog. The communication between two applications could be made from the first application to Watchdog and from Watchdog to the second application.

This solution implies to modify the applications architecture to receive and send messages with the watchdog. Also the watchdog configuration file is modified with some new attributes.

Both these innovative aspects and contributions add a better control of crash situations and process failure. The interface also improves the communication between processes and the Watchdog.

R E F E R E N C E S

[1] Inter Process Communications, (http://users.actcom.co.il/~choo/lupg/tutorials/multi-process/multi-process.html for a tutorial)
[2] SIP- Session Initiation Protocol., http://www.ietf.org/html.charters/sip-charter.html for details
[3] *M. Barr*, "Watchdog Timers" article, Embedded Systems Programming,, pp. 79-80
[4] *M.J. Rochind*: Advanced Unix Programming , Prentice-Hall, Englewood Cliffs
[5] RFC 3261 - Internet Official Protocol Standard for Session Initiation Protocol (SIP), an application-layer control (signaling) protocol for creating, modifying, and terminating sessions with one or more participants. These sessions include Internet telephone calls, multimedia distribution, and multimedia conferences. See details at http://www.faqs.org/rfcs/rfc3261.html
[6] *A.S. Tanenbaum*, Modern Operating Systems,2nd edtition, Prentice Hall
[7] *A.S. Berger*, Embedded Systems Design, CMP Books
[8] *K. Kopper,* The Linux Enterprise Cluster. Build a Highly Available Cluster with Commodity Hardware and Free Software, No Starch Press

[9] *M. Meskes*, Watchdog,: The Linux Software Daemon, Belltown Media Houston, TX.
[10] *Murpy Niall,* “Watchdog Timers,” Embedded Systems Programming, p.112
[11] *J. Santic*, “Watchdog Timer Techniques”, Embedded Systems Programming, p. 58.