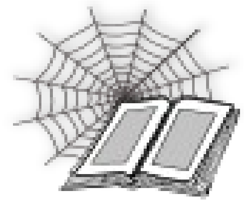
SASSY



ZERO IMPACT NOTIFICATION CHANNEL

Overview

Publication History

Date	Who	What Changes
9 November 2017	Brenton Ross	Initial version.

Copyright © 2009 - 2017 Brenton Ross
This work is licensed under a Creative Commons Attribution-Share Alike 3.0 Unported License.
The software is released under the terms of the GNU General Public License version 3.

Table of Contents

1 Introduction....................4
1.1 Scope....................4
1.2 Overview....................4
1.3 Audience....................4
2 Initial Thoughts....................5
2.1 The Mechanism....................6
2.2 The Solution....................8
Appendix A....................9

1 Introduction

This document provides an overview of the Zero Impact Notification Channel (ZINC). The libzinc library provides client and server code that allows a server to publish its state without the clients being able to affect the server or each other.

1.1 Scope

The document describes the thinking that went into its design.

1.2 Overview

The body of this document is drawn from a conversation on Fedora Forum.

1.3 Audience

Anyone wanting to understand the purpose of libzinc.

2 Initial Thoughts

I have been playing around with trying to implement the actor paradigm in C++. (I think I have something that works - but that is another story). I think this got confused in my mind with a discussion a few days ago about consciousness.

Anyway, the thought is that perhaps it might be useful if programs had an internal object that was advised of the status of the rest of the program. This object would not be involved in the normal processing, except, perhaps, to tweak a few policy type parameters. This object would write its state to a file which could be monitored by other programs (or even a web page). This file would be a snapshot, not a log.

I know this sounds a lot like a logging system, but I think its a bit different - there would be no permanent trace, just the current status. It could expose a lot more about the internals of a program than would be practical for a logging system. I envisage that it would report on things like exceptions, and how busy various part of the program were.

At a previous job we had a system that had processes that ran for months at a time. I often thought it would be nice to see more about how they were going than could be discerned through the exception logging or with the usual UNIX commands. Unfortunately I never had an opportunity to implement that capability, though we did discuss it on more than one occasion.

I can even see where programs would monitor each other and adjust their behaviour according to what the others doing.

> The monitoring itself isn't an issue.
> Its a question what to monitor, what string/message to identify and clarify a specific status for an application.
>
> sea (2015/05/15)

You raise a fair point about what status information to record. I am still considering that but I think it will include exceptions that are caught and handled within the program and the transaction rate of the various parts of the program (where applicable).

I am not intending for this to be part of, or managed by, yet another service. It is more of an ability for any program to monitor how other programs are performing and perhaps adjust their own behaviour. This is likely to be a long way into the future. For now I just want to be able to visualise how programs are "feeling".

2.1 The Mechanism

My immediate concern is how to get the information out of the program. I am thinking that shared memory or a memory mapped file would be the most efficient and is best able to handle the "one writer - many readers" aspect. This then leads to synchronisation issues - I don't want the writer to block waiting for readers as this could be adapted into a denial of service attack. I thought of writing to a temp file and renaming it, but that would defeat much of the performance of a memory mapped file. I also cannot let the reader just read the data without synch as this would probably crash the reader.

> Something to consider, though mostly applicable to scripts, I guess
>
> Some infinite or broken loops, may increase cpu usage incredible fast, even to the point where the kernel has to throttle the cpu speed.
> How'd you catch something like that?
> Because some parts of my brain believe with any C# broken loop, it might just happen the same way...
>
> Saying, how would you identify a 'proper' incremention of the cpu (usage or its temperature as example) compared to a 'falsely' incremention of it?
> How does an external application know, what/if/where/why/when something is called and do its checks accordingly?
> It cant, unless you handle each and every software out there, OR, let that tool use 'template-sheets' to use which hold the data required to identify the checks for its shipping application.
>
> sea (2015/05/30)

I hadn't considered script usage for this, but I think I can see how it might be wrapped in a program so the facility could be used in a script.

Handling broken or crashing programs is a bit out of scope. That is something for the normal system to handle. This is more for the program that seems to be running OK. Of course, there would be nothing to stop someone from reporting things like cpu usage if they wanted to. This library is for providing a mechanism to make those reports visible - it will up to the program designer to decide what to report and how to collect the data.

The issue of understanding what the data means is real enough. For the first iteration I will just provide the ability to give names and descriptions for each value.

I have a scheme that should allow other programs safe access to the data. The data will be written to two regions and a flag will indicate which one should be read from. The flag will be a single bit in the shared memory area so it cannot

be ambiguous. The data will be stable in each region while the flag indicates it is the current region, and for a while after the flag is changed so as to give the reader time to do its reading. This will allow programs like web servers to asynchronously access the data. When the flag is changed there will also be a notification using a pthread condition variable. This will allow other programs, such as GUIs, to wait for changes.

> Sounds a little like JConsole in Java, which has been around for years (and has its own API).

Not quite the same domain. I am thinking of something built into applications rather than something that monitors them from the outside. This should enable them to report more subtle issues in an application specific manner.

> One typical reason we don't trust self-reporting by software is that when/if the program goes wrong, the self-reports are useless, while the externally 'monitors' remain valid.
>
> Yes there really are schema for creating "health" reports from w/in sw - often timestamps of the most recent unit of work, or counts of the various units of work performed. To update we typically require the sw do a lot of internal checks just as a "sanity check" to probabilify the sw is not off in the weeds.

I agree that this approach has some problems detecting when a program has gone "off in the weeds", but that sort of monitoring is not what I am interested in. For that sort of monitoring I was reading recently about a program that monitored the system calls of another program and used a deep learning algorithm to detect when the monitored program had entered an anomalous state.

> That's great, but you can monitor syscalls & libcalls a lot more reliably with the ptrace interface than by any self-reporting. Maybe you want to create a 'wedge' process (like 'time' or 'strace') that keeps some ptrace'd child process call stats in shm, and avoids reliance on the processes and language specific APIs.

As I think I said - I am not interested in that sort of reporting - that is best left to exception logging from within the program or, as you suggest, something that watches what is going at the system level.
To anthropomorphise it a little: It is the difference between asking you "how do you feel" and putting you through a full medical exam. I am only interested in what the program itself considers to be the state - even if that is not an accurate reflection of the real situation !

> Even in a shm system you are going to need find a way to prevent a race condition - process-A reading a structure half written by process-B. You can create global semaphores, or use a very simple CORBA process that just stores/distributes the incoming data and keeps a per section semaphore. There are no totally trivial solutions.

I am approaching this by storing the data twice. The data is written alternating between two buffers with a boolean flag to indicate which is the most recent. The buffers are updated at regular intervals (eg 1 second) which gives the reader at least that long to read the data. I will also have a notification system using pthread_cond_broadcast that will allow the readers to wait for updates.

I am aware that the design will not work for data values that are rapidly updated. The design provides a snapshot of the state, not a stream.

> You are just creating a lower probability of error - not a correct design. You could learn the Dekker algorithm, or see Vanna about a counting semaphore.

The Dekker algorithm is not applicable here as there is an unknown number of reader processes.

There are two approaches to the problem of accessing shared data - one is to lock it so that only one process has access at a time - the other is to detect when a collision has occurred and retry. My solution is of the latter variety.

2.2 The Solution

Hurray - the library is now completed and seems to be working just fine.

I abandoned the idea of using the pthread condition variable for signalling. Instead I am synchronising the readers with the writer using some time information in the shared memory and calls to gettimeofday. The reason was that the necessary mutex gave a rogue reader the ability to block other readers and to stall the thread in the writer. This would not directly affect the rest of the writer, but I thought it would be better to not have any means for readers to influence each other or the writer.

The basis of the design derives from the database world for applications that use a database for storing large documents across many tables. Instead of locking the tables which would seriously impact performance they rely on the fact that each document uses a separate set of database rows and that collisions would normally be quite rare, and do a check to ensure there was no collision and retry if there was one.

Appendix A