MONITORING APPLICATIONS ONCE THEY ARE RELEASED INTO THE USER COMMUNITY*

Sev Binello#, Ted D'Ottavio, Jonathan Laster, Dan Ottavio,
Brookhaven National Laboratory, Upton, New York, USA

Abstract

Once an application is released into the user community, obtaining prompt and high quality information on application usage, applicability and reliability can be a challenge. Most Linux and Solaris applications used at RHIC and associated accelerators have been instrumented so that application and crash information is gathered, stored and forwarded to the appropriate developer for immediate analysis. To support this process, databases were created to track developer and application information. In order to keep these databases relevant, a web based application release procedure was created to collect information and automatically update the database. Additional capabilities have been developed that utilize and expand on the various components of this system to promote communication between developers and users, and to monitor applications. An application feedback feature allows users to instantly communicate with application developers. An application history system records application usage and reliability.

BACKGROUND

Once an application is released into the user community, developers are often unaware of its reliability, applicability or usage. At the Collider Accelerator Department (C-AD) at BNL, the user community consists of operators, physicists and equipment specialists. Prior to the development of the work described here, there was little information available on the reliability, applicability or usage of Controls applications. Crashes went unreported and core dump files were ignored. Furthermore, no automated mechanism was in place to capture and collect information on whether applications met user needs or expectations. Finally, nothing was in place to capture program usage patterns that could be mined for useful information.

APPROACHES

Three systems were developed to address these concerns. A "Crash Utility" system was developed to capture extensive crash information and forward it to developers. A "Send Feedback" system was developed to establish direct communication between application users and developers. An "Application History" system was developed to track application usage and reliability.

Examples of similar capabilities are encountered in commercial software. In Windows, there is the "Online Crash Analysis" (OCA) that generates the ubiquitous "Please tell Microsoft about this problem" message when a Windows application crashes. Many third party applications also typically include a "Send Feedback" feature.

Crash Utility System

The Crash Utility system was developed to collect and promptly forward crash information to application developers. The Crash Utility system consists of four main components: a C library, a "CrashUtility" process, a web-based application release procedure, and a development environment infrastructure.

The main function of the C library is to set up signal handlers that are entered when an application crashes. The main function of the signal handlers is to instantiate the CrashUtility process. It is this latter process that performs most of the crash post processing. This design has two main advantages. It puts few burdens on the application once it crashes, and it allows for most modifications to the Crash Utility system functionality to be performed without requiring the application to be rebuilt.

In order to keep work to a minimum at the time of the crash, static information is collected and prepared when the signal handlers are declared. This includes setting up commands and collecting static information such as process id, start time, console name, process name, and build date.

Once the signal handler is entered its function is twofold. It first executes a "pstack" command to obtain a stack trace. The stack trace provides at least a minimal amount of information and is a convenient means for developers to quickly recognize crashes. It then forks off the separate CrashUtility process.

The function of the CrashUtility process is to store the core file, gather information from the user, collect process information, determine the developers to contact, and send email notification. Core files are compressed and moved to application specific directories. If the application has a graphical user interface, a window is generated to gather additional information from the user. Process information is collected that includes Unix signal name, primary and secondary contacts, time of crash, host machine, display, Unix pid, executable path and name, build date and executable date, start time, version control information, login name, current working directory. Crash information is then formatted into an email and forwarded to a set of developers and managers, as well as the user. Note that it

*Work performed under Contract Number DE-AC02-98CH10886 with the auspices of the U.S. Department of Energy.
sev@bnl.gov

would also be possible to email compressed core files for off-site analysis.

Prompt email notification is an important feature of this process. Not only does it provide the developer with information necessary to help resolve the problem, it also establishes a communication link between the user and the developer.

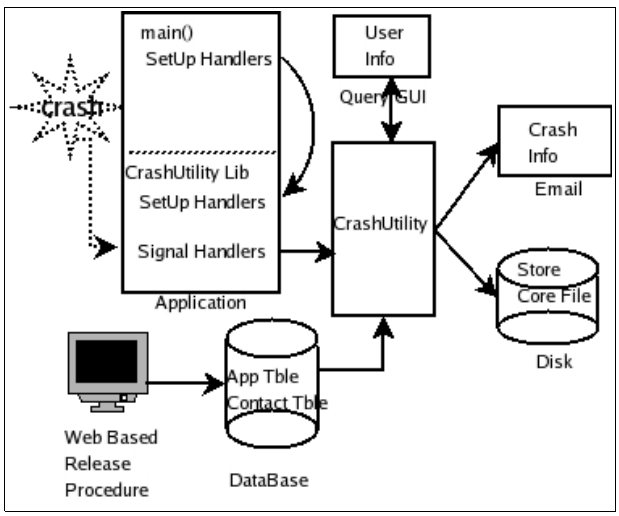

Figure 1: Crash utility system.

To support email notification, a “Diagnostic” database was created that contains application specific information, such as individuals to contact, release dates, links to documentation and comments, as well as developer contact information. In order to maintain this database, a web based application release procedure was created to collect information whenever an application is released to the user community.

All this data, though significant, would still be of limited use if the core file and stack trace did not contain symbolic information. With that in mind, the development environment is configured so that all C++ and C programs are built with the debugger option turned on (i.e. g++ -g), and symbolic information is not stripped from the executable. This allows developers to debug an application's core file as if they had been running the application in the debugger in the first place. The development environment is also configured to embed information in the application during the build process. Information about the kernel, operating system, compiler, application source version, and build date is embedded in the executable.

To control the number of core files, only applications released into the user community are processed in this manner. This default behaviour may be modified by environment variables. Furthermore, scripts are periodically executed to delete old core files.

From a developer's perspective, the requirements are simply: to make a single function call to the Crash Utility library, to adhere to the build and release conventions, and to respond reasonably promptly to crash reports. There are no requirements for the user, only a request for additional information.

Send Feedback System

Various components of the crash utility system have been reused and expanded to facilitate communication between users and developers. One of the most useful features was the inclusion of a "Send Feedback" feature in nearly all applications. This feature provides users with the ability to notify developers about a bug as well as to send comments and requests to developers and managers.

All applications with a graphical user interface are outfitted with a standard menu bar that includes a 'Help' menu. The 'Help' menu contains a 'Send Feedback' button, which is the user's hook to the Send Feedback System. A “sendFeedback” process is instantiated and presents a window to the user. The user can enter their name, email address, priority level of the request, and the feedback message.

An e-mail message is then sent to the user, developer, and management. Further, an entry is made in the department's "Action Please” trouble-tracking system as an item needing attention from the Controls' Group. This allows the department to track the request as if it was made directly from the departmental web-based "Action Please" system.

Application History System

An Application History system was developed to track application usage and reliability. This system records start and stop times, exit status, machine and user name. The goal of this system is to better manage applications. Discovering unreliable applications is obviously important, but also knowing how often the application is used can help determine priorities. Determining key users of an application would be helpful in the event modifications are planned. Knowing who is running an application can be used to notify users when a new version is released.

The Application History system is built around a client/server model in which applications notify the server when they start and exit and pass it relevant information. The server in turn saves this information into the “Application History” database. Communication between clients and server is straight forward. Messages are written to an NFS mount point by the clients and are read by the server. Once the server receives the message, the message contents are sent to the database and the message is removed.

An advantage of this approach is that the communication is asynchronous, so clients can send messages to the server without blocking. Regardless of the responsiveness of the server, clients can always write a message and continue. Another advantage is that it only depends on NFS for it to function correctly.

The overhead incurred due to polling for new messages is slight. Since it is acceptable for messages to reach the database in the order of seconds, the polling time interval is generous. Writing and reading to disk is also not terribly burdensome as the amount of data transferred is quite small, approximately 2K. One distinct drawback of this approach is the inability to detect when a client

receives a SIGKILL (i.e. kill -9). Application History relies on the assumption that an application will write a message when it stops. However, since there is no way to trap a SIGKILL signal, there is no way for a client to write a stop message. To cope with this limitation a separate utility is used to periodically confirm the existence of applications listed in the database as running.

The Application History database is accessible from a web-based interface. The interface presents information in several convenient views: all applications, all currently running applications, and all crashed applications. It also provides search and plotting features.

EXPERIENCE

The "Crash Utility", "Send FeedBack" and "Application History" systems have come on-line at C-AD over a number of years. The Crash Utility has been in place since 2002, followed quickly thereafter by the Send Feedback system, and lastly in 2005, the Application History system.

Email notification of crashes has proved useful to users, developers, and managers alike. A problem with an application is quickly realized. Developers have found that the capture of crash information and storage of core files with debug information has greatly facilitated debugging. This in turn has led to quicker resolutions and increased reliability. The Send Feedback system has increased and facilitated communication between users, developers and management. Operators, especially, have made frequent use of this system. Recording the feedback requests in the Action Please system has led to increased responsiveness to user concerns. The Application History system has provided a means to track application usage and reliability, leading to improved application management.

In summary, the work described in this paper has increased communication, responsiveness and reliability. It has been well received by users, developers and management.