Wrocław, 5.12.2007

Grzegorz Pietrzak (133329) (grzegorz.pietrzak@gmail.com)
Piotr Twaróg (133395) (piotr.twarog@gmail.com)
Grupa: WT/P 11:15-13:00

Projekt: Internet w gospodarce

Monitorowanie dysków twardych z wykorzystaniem technologii S.M.A.R.T. Oprogramowanie sensora monitorującego stan dysku twardego, który powiadamia administratora o niebezpieczeństwie utraty danych

Prowadzący:
dr inż Tomasz Walkowiak

Spis treści

1. Wprowadzenie

1.1. Wstęp

Celem projektu jest stworzenie oprogramowania, które ułatwiłoby administratorowi systemu Linux kontrolę nad sprawnością sprzętu poprzez monitoring stanu dysków twardych oraz powiadamianie na bieżąco o ewentualnych ich awariach. Wykorzystany został program D-Sensor służący do nadzorowania stanu systemu (pamięci, CPU, procesów), używający mechanizmów Java Native Interface do komunikacji z linuksowymi bibliotekami. Główne funkcje D-Sensora to prezentacja wyników pomiarów oraz alarmowanie w przypadku przekroczenia przez dany parametr dozwolonej wartości. W krytycznych chwilach generowane są raporty w formacie XML zawierające wartości będące przyczyną alarmu.

1.2. Krótkie wprowadzenie do S.M.A.R.T.

Monitorowanie dysków twardych odbywa się za pośrednictwem systemu S.M.A.R.T. (ang. Self-Monitoring, Analysis and Reporting Technology). System ten implementowany jest w dyskach ATA (od wersji ATA-3), ATAPI, IDE oraz SCSI-3, aby zwiększyć bezpieczeństwo składowanych danych. Dzięki niemu możliwe jest skuteczne ostrzeganie o zbliżającej się awarii w około 30 do 40% przypadków.

S.M.A.R.T. może być wykorzystywany zarówno do monitorowania, jak i przeprowadzania kilku rodzajów testów. Na potrzeby niniejszego projektu zaadoptowana została jedynie część monitorująca parametry (ze względu na specyfikę działania oprogramowania D-Sensor).

S.M.A.R.T. przechowuje w swoich rejestrach szereg parametrów określających stan dysku, każdy parametr ma swój identyfikator, aktualną wartość (*RAW_VALUE*), wartość znormalizowaną do przedziału 0-100 lub 0-255 (*VALUE*), próg, poniżej którego występuje błąd (*THRESH*) oraz najgorszą uzyskaną dotąd wartość (*WORST*). Kiedy wartość *VALUE* schodzi poniżej progu *THRESH* sygnalizowana jest zbliżająca się awaria. Administrator jest wówczas w stanie zapisać ważne dane lub uruchomić rezerwowy dysk.

1.3. Java Native Interface

Wykorzystywany w projekcie D-Sensor jest aplikacją napisaną w języku Java, w środowisku Fedora Core 6. Monitorowanie systemu możliwe jest w większości przypadków wyłącznie za pośrednictwem systemowych bibliotek C i C++. Powstaje tu poważny problem, gdyż Java to język kompilowany do postaci uruchamianej przez maszynę wirtualną, teoretycznie niezależny od platformy systemowej.

Kwestię kompatybilności Javy z bibliotekami napisanymi w innych językach programowania rozwiązuje (przynajmniej częściowo) JNI - *Java Native Interface*. Interfejs ten pozwala na wywoływanie natywnych (macierzystych, występujących tylko na danej platformie) aplikacji oraz bycie przez nie wywoływanym. Używając JNI ograniczamy znacznie wieloplatformowość oprogramowania, ale zyskujemy dostęp do wywołań systemowych. Standardowo D-Sensor wykorzystuje JNI do obsługi biblioteki statgrab zwracającej statystyki generowane przez poszczególne elementy Linuxa. W niniejszym projekcie JNI posłuży jako interfejs pomiędzy Javą a skompilowanym do postaci pliku .so programem monitorującym dyski twarde.

2. Wybór narzędzia do monitorowania dysków

Pierwszym krokiem do realizacji projektu było znalezienie odpowiedniego oprogramowania służącego do obsługi systemu S.M.A.R.T. Kryterium poszukiwań były otwartość licencji oraz możliwość kompilacji do postaci biblioteki łączonej dynamicznie (z rozszerzeniem .so). Wymagania te zostały spełnione przez dwie aplikacje:

- **smartsuite** – niewielki, nie rozwijany już od 2001 program, którego źródła składają się z zaledwie kilku plików. Jego największą wadą jest niewielka liczba typów obsługiwanych dysków.
- **smartmontools** – duży, wieloplatformowy pakiet będący integralną częścią większości dystrybucji Linuxa. Stanowi on rozszerzenie opisanego w poprzednim punkcje *smartsuite*. Różnice polegają na wprowadzeniu pełnej obsługi Serial ATA i SCSI, a także wsparciu dla szerokiej gamy dysków. Smartmontools napisany jest w C++, jego kod źródłowy jest bardzo rozbudowany.

W projekcie został użyty *smartmontools* – pomimo dużego skomplikowania kodu stanowi on już niemal standard w dziedzinie monitoringu poprzez S.M.A.R.T. Działa na większości platform systemowych (w tym pod Windows). Poniżej znajduje się przykład działania zawartego w pakiecie programu *smartctl*:

```
ID# ATTRIBUTE_NAME          FLAG     VALUE WORST THRESH TYPE     WHEN_FAILED RAW_VALUE
  1 Raw_Read_Error_Rate     0x0029   100   253   020    Pre-fail     -       0
  3 Spin_Up_Time            0x0027   074   074   020    Pre-fail     -       3294
  4 Start_Stop_Count        0x0032   100   100   008    Old_age      -       32
  5 Reallocated_Sector_Ct   0x0033   001   001   020    Pre-fail FAILING_NOW 499
  7 Seek_Error_Rate         0x000b   100   001   023    Pre-fail In_the_past 0
  (...)
```

W podanym przykładzie S.M.A.R.T. sygnalizuje awarię dysku ze względu na zbyt dużą wartość parametru „reallocated system count”, oznaczającego liczbę uszkodzonego sektorów, które zostały przeniesione do obszarów zapasowych. Najlepszą znormalizowaną wartością parametru jest 100, wartością aktualną – 1, zaś progiem wystąpienia błędu – 20.

D-Sensor, jako aplikacja monitorująca, wykorzystuje jedynie kolumny „VALUE” i „RAW_VALUE” powyższej tabeli wynikowej. Administrator konfigurujący moduł S.M.A.R.T. powinien znać wartości progów („THRESH”) dla zarządzanych przez siebie dysków, może otrzymać je np. za pośrednictwem programu *smartctl.*

Duże problemy sprawiają różnice w implementacji S.M.A.R.T. pomiędzy dyskami różnych producentów. Większość dodatkowych lub specyficznych dla danej firmy funkcjonalności wymaga monitorowania i przechowywania wyników pomiarów w osobnych parametrach. W projekcie kwestia mnogości implementacji S.M.A.R.T. została rozwiązana poprzez wybór uniwersalnego zestawu najważniejszych parametrów dysku. Jeżeli parametr nie jest obsługiwany, jego VALUE przyjmuje wartość 256 (czyli większą o 1 od największej możliwej), dzięki czemu nigdy nie wywoła fałszywego alarmu w D-Sensor (przy założeniu prawidłowej konfiguracji).

3. Moduł C++ do Java Native Interface

3.1. Konstrukcja modułu

Aby móc korzystać poprzez JNI z funkcji zawartych w *smartmontools* należy zbudować dynamicznie dołączaną bibliotekę, której funkcje wywoływałaby wyznaczona klasa za pośrednictwem natywnych metod.

Wymagane jest utworzenie klasy *SmartStats*, odpowiedzialnej za odczyt i przechowywanie wyników, posiadającej metody *readSmartStatistics()* do odczytu i *printSmartStatistics()* do wypisania danych, a także natywną metodę *getSmartStats()*, której definicja znajduje się w bibliotece .so. Poniżej znajduje się (nieco skrócona) zawartość pliku *SmartStats.java*:

```
package eu.deserec.pwr.itemAgent.linux.systemStats.libDiskStats;
import java.io.Serializable;

public class SmartStats implements Serializable
{
	private static final long serialVersionUID = 1L;
	private native void getSmartStats();
	public void readSmartStatistics()
	{
		getSmartStats();
	}

	public void printSmartStatistics()
	{
		...
	}

	...
}
```

Po utworzeniu klasy w Javie należy wygenerować plik nagłówkowy dla języka C. Plik .h będzie zawierał deklaracje natywnych funkcji, wywołanych przez Javę. Istnieją dwa sposoby automatycznej generacji nagłówka – albo poprzez bezpośrednie wywołanie w linii poleceń „javah” z parametrem określającym pełną nazwę rozpatrywanej klasy (w naszym przypadku *eu.deserec. ... .SmartStats*), albo poprzez odpowiednie skonfigurowanie pliku build.xml i wywołanie „ant javah”. Drugi ze sposobów jest łatwiejszy i bardziej elastyczny, wystarczy dodać do build.xml następujące linie:

```
<property name="libDiskStats.pkg" value="${systemStats.pkg}.libDiskStats"/>

<target name="javah" depends="init">
	<javah  destdir="${jni_src.dir}" verbose="${talk.var}" >
		<class name="${libDiskStats.pkg}.SmartStats"/>
		...
	</javah>
</target>
```

Do późniejszego skompilowania pliku nagłówkowego będzie potrzebny plik jni.h znajdujący się w katalogu zawierającym JDK Javy. Zawiera on definicje stałych JNIEXPORT i JNICALL oraz treść struktury JNIEnv.

Otrzymany po wywołaniu w konsoli „ant run” plik *eu_deserec_pwr_itemAgent_linux_systemStats_libDiskStats_SmartStats.h* (jego długa nazwa odzwierciedla nazwę klasy w Javie) ma następującą postać:

```
#include <jni.h>
/* Header for class eu_deserec_pwr_itemAgent_linux_systemStats_libDiskStats_SmartStats
*/
#ifndef _Included_eu_deserec_pwr_itemAgent_linux_systemStats_libDiskStats_SmartStats|
#define _Included_eu_deserec_pwr_itemAgent_linux_systemStats_libDiskStats_SmartStats
#ifdef __cplusplus
extern "C" {
#endif
#undef
eu_deserec_pwr_itemAgent_linux_systemStats_libDiskStats_SmartStats_serialVersionUID
#define
eu_deserec_pwr_itemAgent_linux_systemStats_libDiskStats_SmartStats_serialVersionUID
1LL
/* Class:     eu_deserec_pwr_itemAgent_linux_systemStats_libDiskStats_SmartStats
 * Method:    getSmartStats
 * Signature: ()V
 */
JNIEXPORT void JNICALL
Java_eu_deserec_pwr_itemAgent_linux_systemStats_libDiskStats_SmartStats_getSmartStats
  (JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
#endif
```

Na podstawie pliku nagłówkowego należy napisać plik *eu_deserec_pwr_itemAgent_linux_systemStats_libDiskStats_SmartStats.cpp* (używamy C++ ze względu na kompatybilność ze *smartmontools*, stanowi to spory problem ze względu na słabe wsparcie JNI dla C++). Zalążka pliku .cpp nie da się wygenerować automatycznie, trzeba utworzyć go ręcznie używając deklaracji zawartych w nagłówku. Plik .cpp powinien zawierać odczyt parametrów S.M.A.R.T. i ich przypisanie do zmiennych klasy SmartStats.

Głównym ogniwem budowanej biblioteki są elementy *smartmontools*. Tworzony plik .cpp zawiera funkcje napisane na wzór odpowiedników ze *smartmon.cpp*. Kompilacja w postaci biblioteki wymusiła trzy uproszczenia:

- brak obsługi dysków SCSI (nieco odmienny sposób działania, brak możliwości testowania modułu na rzeczywistym dysku),
- nazwę monitorowanego urządzenia (np. /dev/sda lub /dev/hda) należy przed uruchomieniem D-Sensora wpisać do pliku smart.conf znajdującego się w głównym katalogu aplikacji. Jeśli plik nie zostanie odnaleziony wówczas domyślnie zostanie przyjęte urządzenie /dev/sda (czyli np. najczęściej dziś spotykany interfejs SerialATA z kontrolerem RAID)
- brak obsługi dysków firmy Samsung (nie da się automatycznie wykryć ich typu, a wymagają odmiennego traktowania).

Funkcja *smart_init()* inicjalizuje odczyt ze S.M.A.R.T. otwierając deskryptor odpowiedniego urządzenia (wymagane prawa administratora!). Funkcja *get_ata_values()* odczytuje parametry dysku do zmiennych *wartosci* (struktura typu *ata_smart_values*) i *progi* (struktura typu *ata_smart_thresholds_pvt*). Definicja natywnej funkcji *getSmartStats()* zawiera przypisanie odpowiednich wartości z tych struktur do zmiennych publicznych klasy *SmartStats*.

3.2. Kompilacja modułu

Kompilacja biblioteki (nazwanej *libJNIDiskStats.so*) odbywa się poprzez Makefile, po wpisaniu komendy „make disk” w katalogu zawierającym pliki .c związane z JNI:

```
SOURCES1 =  eu_deserec_pwr_itemAgent_linux_systemStats_libDiskStats_SmartStats.cpp \
            smt/atacmdnames.cpp \
            smt/atacmds.cpp     \
            smt/ataprint.cpp    \
            smt/knowndrives.cpp \
            smt/utility.cpp \
            smt/scsiata.cpp \
            smt/scsicmds.cpp \
            smt/os_linux.cpp


disk: $(SOURCES1)
     gcc -lstdc++ -Wall -I/usr/lib/jvm/java-6-sun-1.6.0.00/include
-I/usr/lib/jvm/java-6-sun-1.6.0.00/include/linux/ $(SOURCES1) -shared
-o ../../../lib/jni/libJNIDiskStat.so
```

Skompilowana biblioteka trafia do folderu lib/jni, skąd można ją załadować z poziomu języka Java przy pomocy instrukcji:

```
static { System.loadLibrary("JNIDiskStat");}
```

3.3. Dodanie modułu do D-Sensor

Integracja modułu SmartStats z programem D-Sensor wymaga stworzenia kilku nowych plików oraz zmodyfikowaniu istniejących. Przygotowany w poprzednim punkcie plik *SmartStats.java* należy skopiować do katalogu *src/java/eu/deserec/pwr/itemAgent/linux/systemStats/libDiskStats*. Aby statystyki dotyczące dysku były monitorowane przez D-Sensor wymagana jest odpowiednia klasa dziedzicząca po *StatisticsCollector.* W naszym przypadku jest to klasa *SmartStatisticsCollector* zawierająca metodę *run()*, w której do ogólnej puli obserwowanych wartości dodawane są kolejne parametry S.M.A.R.T.:

```
public class SmartStatisticsCollector extends StatisticsCollector {
   private SmartStats smartStats;

   public SmartStatisticsCollector() {
       super();
       this.smartStats = new SmartStats();
   }
   @Override
   public void run() {
      (...)
      message.addStatistic("smart.raw_read_error_rate",
smartStats.raw_read_error_rate);
      message.addStatistic("smart.raw_read_error_rate_raw",
smartStats.raw_read_error_rate_raw);
      (...)
   }
}
```

Do pliku *StatisticsCollector.java* trzeba dodać linijkę kodu ładującą skompilowaną w poprzednim punkcie bibliotekę .so. Gdy collector jest już gotowy, w pliku *StatsDSensor.java* musi znaleźć się jego inicjalizacja. Do pełnej funkcjonalności brakuje już tylko umieszczenia w *CollectorType.java* typu „SmartStats".

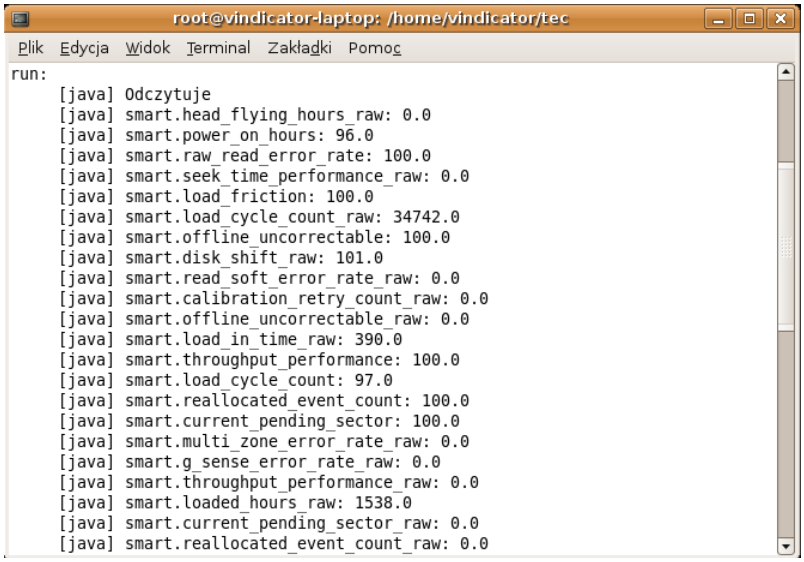
Informacje o dysku mogą mieć charakter informacyjny lub ostrzegawczy. W trybie ostrzegawczym otrzymujemy możliwość zakładania „pułapek" na monitorowane parametry, dzięki czemu administrator jest natychmiast informowany o przekroczeniu dozwolonych wartości zdefiniowanych odczytów. Zmiana trybu odbywa się poprzez wywołanie *message.setSeverity("information")* lub *message.setSeverity("warning")* w pliku *SmartStatisticsCollector.java*.

4. Uruchomienie projektu

Do poprawnej kompilacji projektu wymagana jest Java JDK w wersji 6 (należy uważać tu na kompilator używany przez Eclipse, gdyż jest on w wersji 5 bez względu na ustawienia systemowe). W przypadku istnieniu obok siebie w systemie kilku wersji Javy trzeba pamiętać o ustawieniu najnowszej jako bieżącej (przy pomocy polecenia „alternatives --config java" wywołanego również dla *javac* i *javah*). D-Sensor standardowo wykorzystuje biblioteki JAXB (Java Architecutre for XML Binding), niekompatybilne z ich odpowiednikami z Java 6. Aby umożliwić kompilację należy skopiować pliki „ja*.jar" z katalogu „lib/" do „$JAVA_HOME/jre/lib/endorsed/".

D-Sensor z dołączonym modułem S.M.A.R.T. został przetestowany w dystrybucjach Fedora Core 6, Ubuntu 7.04 i Ubuntu 7.10. Główny problem stanowiło ustawienie odpowiednich ścieżek do katalogu zawierającego JDK Javy.

Kompilacja i uruchomienie programu odbywa się za pomocą komend „ant compile" i „ant run". Konfiguracja zawarta jest w pliku testConfig.xml. Poniżej znajduje się przykład, w którym S.M.A.R.T. działa w trybie informacyjnym, wyświetlając na ekranie co 2 sekundy wszystkie obserwowane parametry dysku:



5. Wnioski

Realizacja projektu zakończyła się sukcesem, wszystkie założenia projektowe zostały spełnione. Zgodnie z przewidywaniami najwięcej problemów sprawiła obsługa JNI, głównie ze względu na słabą dokumentację i użycie do budowy biblioteki C++ zamiast standardowego C. Dużą barierę stanowił również nieudokumentowany i pozbawiony komentarzy kod aplikacji D-Sensor.

Dodawanie nowych rodzajów statystyk do D-Sensor nie jest trudne, ale w niemal każdym przypadku wymaga współpracy Javy z kodem napisanym w C. W rezultacie jesteśmy zmuszeni do konstruowania skomplikowanych rozwiązań, ściśle uzależnionych od platformy. Znacznie bardziej zasadne byłoby oparcie aplikacji monitorującej system o język C/C++, co pozwoliłoby na wkompilowanie w nią dodatkowej funkcjonalności i co za tym idzie – rezygnację z nadmiarowych, trudnych do napisania bibliotek łączonych dynamicznie.

6. Linki do stron z opisem S.M.A.R.T.

Do wersji elektronicznej dołączamy linki do stron z opisem parametrów S.M.A.R.T.:

- http://www.ariolic.com/activesmart/smart-attributes/
- http://forum.purepc.pl/
- http://forum.infojama.pl/