Runtime Monitoring & Issue Tracking

http://d3s.mff.cuni.cz

Pavel Parízek
parizek@d3s.mff.cuni.cz
Runtime monitoring
Runtime monitoring

• Goals
  ▪ Recording information about program behavior
  ▪ Notification about specific important events

• Information: performance, security, exceptions

• Target domain: long-running programs
  ▪ Application servers (JBoss, Tomcat, WebSphere, ...)
  ▪ Network servers and daemons (Apache, Sendmail)

• Alternative name: tracing
Basic approaches

- Manual implementation of logging commands
- Using tools for automated runtime monitoring
Tools

- Unix-like platforms
  - Syslog, strace, ltrace, DTrace
- Java ecosystem
  - Log4j 2, Java Logging API, JConsole, JVM TI
- Windows/.NET
  - log4net, NLog, Process Explorer

- Events: custom messages, system calls, library calls
- Output: text log files (off-line inspection), GUI
Log4j

- Popular logging framework for Java platform
  - [http://logging.apache.org/log4j/2.x/](http://logging.apache.org/log4j/2.x/)

- Features
  - Hierarchy of loggers based on class names
  - Filtering messages based on logging levels
  - Dynamically updateable configuration (XML)
  - Multiple output destinations (console, file)
  - Formatting log messages (printf-style, HTML)
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

// get a Logger object with a particular name
Logger logger = LogManager.getLogger("cz.cuni.mff");

logger.warn("Running out of disk space");
...
logger.error("File {} not found", f.getName());
...
logger.info("Something normal happened");
Using Log4j

- **Levels**
  - `TRACE < DEBUG < INFO < WARN < ERROR < FATAL`

- **Logger objects**
  - Identified by logical names (e.g., Java class names)
  - They make a hierarchy based on the name prefixes
    - Logger named “cz.cuni” is a parent for the Logger “cz.cuni.mff”
    - Inheriting configuration (levels, appenders, formatting pattern)

- **Configuration:** XML, programmatic
  - Default file name `log4j2.xml` *(must be on classpath)*
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
    <Appenders>
        <Console name="konzole" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss} %-5level %c{36} - %m%n"/>
        </Console>
        <File name="logfile" fileName="test.log">
            <PatternLayout pattern="%d{HH:mm:ss} %-5level %c{36} - %m%n"/>
        </File>
    </Appenders>
    <Loggers>
        <Logger name="cz.cuni.mff" level="info">
            <AppenderRef ref="konzole"/>
        </Logger>
        <Root level="error">
            <AppenderRef ref="logfile"/>
        </Root>
    </Loggers>
</Configuration>
Appenders

- Responsible for writing log messages to actual target destinations

- Supported targets
  - Console (stdout, stderr)
  - File (buffered, appending)
  - Database (via JDBC)
  - SMTP (sending emails)
  - Network socket (TCP, UDP)
  - Unix/Linux syslog service
Layout

• Purpose: formatting messages

• Available layouts
  • Pattern
    • `%m`  // message text
    • `%n`  // line separator
    • `%5level`  // level, justified to the right, width five chars
    • `%d{HH:mm:ss}`  // current datetime with pattern
    • `%c{20}`  // logger name with the maximal length
    • `%C %M %L`  // class name, method name, line number
    • `%t`  // thread name
  • HTML, XML, Syslog
public Object doSomething(int arg1) {
    logger.entry(arg1);
    try {
        ...
        Object res = ...
    }
    catch (Exception ex) {
        logger.catching(ex)
    }
    logger.exit(res);
}
Task 1

• Download Log4j from the web (JAR files)
  ▪ [http://logging.apache.org/log4j/2.x/](http://logging.apache.org/log4j/2.x/)

• Write simple program in Java
  ▪ You can also take some existing program (anywhere)

• Try important features of Log4j
  ▪ Use several Loggers
  ▪ Different log levels
  ▪ Configuration (XML)
  ▪ Tracing control flow

• Check the output (console, log files)
Log4j: other features

- Filtering messages
  - markers, regular expression, time

- Automatic reconfiguration
  - if you update the XML configuration file at runtime
Syslog

- Standard logging framework for Unix-like systems

- Service
  - Collecting messages from different sources (applications)
  - Writing received messages to various output destinations
    - log files (/var/log), another computer over network
  - Configuration: /etc/syslog.conf, /etc/rsyslog.conf
  - Log rotation: /var/log/messages, /var/log/messages.1, ...

- Protocol
  - Format of data exchanged between applications and the service
  - Message: content (plaintext, < 1024 bytes), priority
  - Supported priorities (low to high)
    - debug, info, notice, warning, error, critical, alert, emergency
  - Definition: RFC 3164, 3195
Configuration: example

Sendmail
Apache
MySQL

Syslogd

/var/log/mail.log
/var/log/mysql.log
/var/log/httpd/httpd.log
Syslog API: example

```
#include <syslog.h>

openlog("myprog", LOG_CONS | LOG_PID, LOG_USER);

syslog(LOG_NOTICE, "Program runs for %d hours", 2);
syslog(LOG_ERROR, "File %s does not exist", fname);

closelog();
```
strace

- Tool for monitoring interactions with the operating system kernel
  - System calls performed by the given program
  - Signals received by the given program from OS
- Available for Unix-like platforms

- Usage: `strace <program>`
  - Attaching to a running process: `strace -p <pid>`

- Output: list of system calls and signals
  
  ```
  open("/etc/passwd", O_RDONLY) = 3
  open("/etc/passwords", O_RDONLY) = -1 ENOENT (No such file)
  ```
Task 2

• Try using
  - `strace (syscalls)`
  - `ltrace (libraries)`

• Check output
JConsole

- Available in Oracle JDK

- Key features
  - Provides useful information
    - CPU usage, memory consumption, threads
  - Nice graphical interface
  - Connection to remote JVM

- How to run it: `jconsole`

- Live demo
Windows Sysinternals

- Process Explorer
  - Displays information about running processes

- Process Monitor
  - Displays some live (real-time) process activity
Issue tracking
Issue tracking systems

- Typically part of a project management system
  - Example: http://sourceforge.net/

- Popular systems
  - Bugzilla, Trac

- Components
  - Some database of known bugs
  - User interface (WWW, desktop)
Bug characteristics

- Time of reporting
- Product (module)
- Version of the product
- **Severity of the bug**
  - blocker, critical, major, normal, minor, enhancement
- Platform (OS, HW, SW)
- Textual comments
- **Current status**
  - new, unconfirmed, assigned, fixed, wontfix, resolved
- Assigned to
  - Who should fix the bug
Lifecycle of a bug

Possible resolutions:
- FIXED
- DUPLICATE
- WONTFIX
- WORKSFORME
- INVALID

Figure taken from http://www.bugzilla.org/docs
Bugzilla

- Web-based tool
  - [http://www.bugzilla.org](http://www.bugzilla.org)

- SW requirements
  - Database (MySQL, PostgreSQL)
  - Perl 5 with specific modules
  - Web server (e.g., Apache httpd)

- Features
  - Advanced queries
    - Boolean operators (and, or, not)
  - Saved search
  - Cloning of bugs
Task 3

- Try out Bugzilla
  - Entering new bug reports
  - Search for existing bugs
  - Changing status of a bug

- Example: [https://bugzilla.mozilla.org](https://bugzilla.mozilla.org)

- Test installation
    - You can play with that freely (reporting new bugs, etc)
    - Requires personal account (email address, password)
Trac

- Project management system
  - http://trac.edgewall.org/

- Features
  - Tracking issues (bugs, feature requests)
  - Good integration with version control
    - Supported tools: Subversion, Mercurial, Git
    - Links from bug reports to source code files
  - Source code browser (version control)
  - Wiki pages (e.g., for documentation)
Other links

• Syslog

• log4net

• DTrace
  ▪ [http://dtrace.org/blogs/about/](http://dtrace.org/blogs/about/)

• JConsole
  ▪ [http://docs.oracle.com/javase/7/docs/technotes/guides/management/jconsole.html](http://docs.oracle.com/javase/7/docs/technotes/guides/management/jconsole.html)
Homework

• Assignment
  - http://d3s.mff.cuni.cz/~parizek/teaching/sdt/

• Deadline